NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ANALYSIS OF BINARY XML SUITABILITY FOR NATO TACTICAL MESSAGING

by

Matthew E. Bayer

September 2005

Thesis Advisor:	Don Brutzman
Co-Advisor:	Terry Norbraten
Second Reader:	Don McGregor

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Analysis of Binary XML Suitability for NATO Tactical Messaging		**5. FUNDING NUMBERS**	
6. AUTHOR Matthew E. Bayer			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME AND ADDRESS		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT

The ability to efficiently transfer information among tactical systems is essential for network-centric operations. However, maintaining interoperability among heterogeneous networks and applications is a challenging issue, especially for large enterprises such as the US Department of Defense and NATO. Each of these organizations maintain extensive communication networks of tactical systems that process and manage all types of data. Additional complexity is added when considering that many systems are built with a variety of proprietary or legacy data formats. Establishing and maintaining interoperability is difficult.

Using XML, many interoperability issues can now be successfully addressed. XML provides a self-describing way to effectively structure information that can be applied to compose diverse tactical communications. However, XML is inefficient for network transmission since it uses a text-based format which can consume more memory (and thus more bandwidth) than binary equivalents. In addition, parsing text-based documents is slow and computationally expensive. One potential solution is to use GZIP to reduce the file size before transmission. Unfortunately, this solution has limitations since it often provides suboptimal compression and also requires additional processing time when extracting data. Recent standardization efforts have identified promising new encodings for XML that use binary representations to reduce parsing time, memory size, and bandwidth requirements.

This thesis surveys conversion of NATO tactical data link information into an XML format for distribution to command and control centers. General benefits and tradeoffs are then considered for applying binary XML encoding to that data. This thesis also examines work done by the World Wide Web Consortium in examining common use cases and developing the requirements needed for a binary XML encoding. The performance of two specific implementations, XML Schema based Binary Compression (XSBC) and Fast Infoset (FI), are compared with GZIP. XML files of varying sizes are encoded in binary form, then compression ratios and parsing times are compared and analyzed. Initial results are excellent and further work is recommended.

14. SUBJECT TERMS: XML Schema Based Compression (XSBC), Extensible Mark-up Language (XML), Tactical Data Links, Binary XML, Fast Infoset			**15. NUMBER OF PAGES** 129
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ANALYSIS OF BINARY XML SUITABILITY FOR NATO TACTICAL MESSAGING

Matthew E. Bayer
Ensign, United States Navy
B.S., United States Naval Academy, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY (C3)

from the

NAVAL POSTGRADUATE SCHOOL
September 2005

Author: Matthew E. Bayer

Approved by: Don Brutzman
Thesis Advisor

Terry Norbraten
Co-Advisor

Don McGregor
Second Reader

Dan Boger
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The ability to efficiently transfer information among tactical systems is essential for network-centric operations. However, maintaining interoperability among heterogeneous networks and applications is a challenging issue, especially for large enterprises such as the US Department of Defense and NATO. Each of these organizations maintain extensive communication networks of tactical systems that process and manage all types of data. Additional complexity is added when considering that many systems are built with a variety of proprietary or legacy data formats. Establishing and maintaining interoperability is difficult.

Using XML, many interoperability issues can now be successfully addressed. XML provides a self-describing way to effectively structure information that can be applied to compose diverse tactical communications. However, XML is inefficient for network transmission since it uses a text-based format which can consume more memory (and thus more bandwidth) than binary equivalents. In addition, parsing text-based documents is slow and computationally expensive. One potential solution is to use GZIP to reduce the file size before transmission. Unfortunately, this solution has limitations since it often provides suboptimal compression and also requires additional processing time when extracting data. Recent standardization efforts have identified promising new encodings for XML that use binary representations to reduce parsing time, memory size, and bandwidth requirements.

This thesis surveys conversion of NATO tactical data link information into an XML format for distribution to command and control centers. General benefits and tradeoffs are then considered for applying binary XML encoding to that data. This thesis also examines work done by the World Wide Web Consortium in examining common use cases and developing the requirements needed for a binary XML encoding. The performance of two specific implementations, XML Schema based Binary Compression (XSBC) and Fast Infoset (FI), are compared with GZIP. XML files of varying sizes are encoded in binary form, then compression ratios and parsing times are compared and analyzed. Initial results are excellent and further work is recommended.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS

AUV	Autonomous Underwater Vehicle
API	Application Program Interface
ASN.1	Abstract Syntax Notation One
BPS	Bits Per Second
C4I	Command, Control, Communications, Computer's and Intelligence
CAOC	Combined Air Operations Center
COTS	Commercial Off-the-Shelf
CVS	Concurrent Versioning System
DFI/DUI	Data Field Identifier / Data Usage Identifier
DIS	Distributive Interactive Simulation
DoD	Department of Defense
DOM	Document Object Model
DTD	Document Type Definition
FEC	Forward Error Correction
GZIP	GNU Zip
ICC	Integrated Command and Control
IDE	Integrated Development Environment
IP	Internet Protocol
ISO	International Standards Organization
LOS	Line of Sight
KB	Kilobytes
MB	Megabytes

Mbs	Megabits Per Second
MD5	Message-Digest Algorithm 5
MOVES	Modeling, Virtual Environments, and Simulation
NATO	North Atlantic Treaty Organization
NC3A	NATO Consultation, Command and Control Agency
NIRIS	Networked Interoperable Real-time Information Services
NPS	Naval Postgraduate School
NVE	Networked Virtual Environments
SA	Situational Awareness
SAX	Simple API for XML
SHAPE	Supreme Headquarters Allied Powers Europe
StAX	Streaming API for XML
STANAG	NATO Standardization Agreement
STGP	Shared Tactical Ground Picture
TCP	Transmission Control Protocol
TADIL	Tactical Digital Information Link
TDL	Tactical Data Links
TIDE	NATO web service that stands for: Transforming technology towards Information superiority, Decision superiority, and Execution Superiority
TITO	Tracks in, TDL out, or, TDL in Track out, or TDL in, TDL out
TIXO	TDL in, XML out
UDP	User Datagram Protocol
UTF-8	Unicode Transformation Format-8

VMF	Variable Message Format
VPN	Virtual Private Network
VRML	Virtual Reality Modeling Language
W3C	World Wide Web Consortium
WISE	A Java GUI portal displaying web-based services and live Link data from the net
X3D	Extensible 3D
XFSP	Cross Format Schema Protocol
XITO	XML in, Tactical Data out
XML	Extensible Markup Language
XSBC	XML Schema-based Binary Compression
XSL	Extensible Style Language
XSLT	Extensible Style Language Transformation
XLTF	Extensible Light Track Format

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Dr. Don Brutzman for his encouragement, patience, and insight into the problems at hand. I am also very thankful for his willingness to advise a young and inexperienced Ensign. His vision and enthusiasm for his work is truly contagious.

I would also like to thank Terry Norbraten for his patience in answering my many questions of both great and small importance. He also accompanied me on my research trip to NC3A and the credit for its subsequent success belongs to him. His excitement towards the project was a great encouragement to me.

Finally, I must thank Jesus Christ who ultimately enabled me to complete this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Maintaining interoperability among heterogeneous networks and data processing systems has proven to be a challenging issue. However, with the advent of the Extensible Mark-Up Language (XML), many of these issues are being resolved as it provides the ability to describe almost any data. However, XML uses a textual format that is expensive to parse and has a large memory footprint. This is because text representations in memory are often larger than a binary equivalent.

The need for a more efficient encoding of XML became apparent. One implementation of this concept, known as XML Schema based Binary Compression (XSBC), was the result of the work of an NPS thesis student seeking to define dynamic protocols used in networked virtual environments. XSBC tokenizes tags and replaces them with integer values. It then tries to reduce the data within the tag to the simplest possible data type while retaining the data's original value. This process coupled with GZIP produces a file that can be less than 10% of its original file size and is faster to parse.

The North Atlantic Treaty Organization (NATO) has performed work to convert tactical link data into an XML format for distribution to command and control centers. The data format transmitted in the tactical environment is the same, but command centers convert the data into XML using software known as the Networked Interoperable Real-time Information Services (NIRIS). NIRIS fuses data to create a common operating picture and has also been modified to act as a server for various web services. With this architecture, a simple web application contacts NIRIS via a secure VPN over the internet and is able to graphically display TDL data or perform whatever task the web application requires.

Finally, this work details the performance characteristics of XSBC as compared to Sun Microsystems' Fast Infoset. The resulting performance shows that XSBC provides superior compression and comparable parse time (within a factor of 3 unoptimized) for files under 500 kb. For larger files XSBC's parsing time begins to grow much more rapidly than Fast Infoset's parse times, indicating that Fast Infoset is a better solution for those cases.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

The United States military is increasingly integrating network technology into its operations. The value of information sharing as well as its feasibility is clearly displayed by the Internet. This provided an impetus for the Network Centric Warfare doctrine which states that "a robustly networked force improves information sharing [which] enhances the quality of information and shared situational awareness [to] dramatically increase mission effectiveness" (Alberts, Garstka, & Stein, 1999).

However, current forces are operating with older equipment whose requirements during design were not influenced by a net-centric doctrine. As a result, many information systems are "stovepiped" meaning they perform their tasks well but are isolated from communicating with other systems due to propriety standards. The development and construction for a system in the military is often contracted out and therefore various systems may be designed and manufactured by several different companies. These interoperability concerns are amplified when working in a joint or coalition environment where systems may be manufactured in different countries.

However, some of these obstacles can be overcome through modeling data using the Extensible Markup Language (XML). This relatively new technology allows one to model virtually any type of data. Once the data from the stovepiped systems is in this common, industry standard format it can be collected and fused for a more robust information sharing capability. The design goals of XML include human readability and ease of use which translates into creating, editing, and transporting XML as text. While this makes developing XML easier, it cause problems in other areas. The most significant drawbacks brought on by a text based format are the large memory footprint and parsing cost. Many XML files are very large text files whose memory footprint is much larger than that of a binary equivalent. Though the data may be interoperable it is not efficient for network transportation in this form. This problem is exacerbated by the limited amount of bandwidth available in the tactical environment. Also, parsing text has a higher computational cost than parsing a more simple data type. This is a concern for

servers running web services that process a large volume of XML documents. A new binary encoding for XML is needed to continue to reap the data modeling benefits of XML while making it cost effective in terms of network and computational resources.

B. MOTIVATION

The recent massive infusion of information technology into military operations is an attempt to gain a tactical advantage through information superiority. The ability to consistently maintain better situational awareness than the enemy through the efficient gathering, fusing, and dissemination of information allows one to iterate through the decision cycle faster than the enemy and keep his operations reactionary, and denying him the ability to take the initiative. Simply put, information superiority seeks to get the right information to the right people at the right time.

The NATO Consultation, Command and Control Agency (NC3A), seeking to use XML to gain information superiority, has already completely converted data from two major tactical data links into an XML format (Howland, Dr. Paul E., 2004). This is a large step towards network centric warfare through communication system interoperability. This is especially necessary for an organization comprised of 26 different countries each complete with its own military equipment and communication protocols and standards. With the XML-ized data in place, however, the issues mentioned above arose. Knowing that the MOVES Institute had developed a working solution NC3A contacted MOVES seeking a binary XML solution. In a collaborative effort, the binary XML solution was successfully integrated into the ongoing work at NC3A.

C. OBJECTIVES

This thesis seeks to assess where XML is a viable solution to implementing network centric doctrine into tactical data links. XML has been used commercially to create and maintain interoperability among dissimilar systems and to model multiple types of data. XML is extensible which provides room for adaptation under frequently

changing requirements. Associated standards provide support for security via validation against an XML schema, digital signatures, and encryption.

This thesis also examines the standardization efforts toward a binary XML solution by the W3C, and examines whether the solution developed at NPS meets the characterization requirements. Currently, no candidate solution has been officially considered, much less a standard chosen since additional future integration and testing is necessary.

Finally, this work assess the performance characteristics of XSBC as compared to Fast Infoset and determine what use cases might derive the most benefit from either solution. Parsing costs and compression ratios are mutually exclusive, which suggests certain scenarios benefit more from other particular solutions.

D. THESIS ORGANIZATION

Six chapters comprise this work

- **Chapter I-Introduction.** This chapter seeks to give a brief overview of the topics discussed as well as the motivation behind the work.
- **Chapter II-Related Work and Background.** This chapter describes the technologies that lay the foundation for the XML as well as two binary XML solutions.
- **Chapter III-Evolution of Binary XML.** This chapter follows the standardization process of the W3C from the beginning workshop to current recommendation for a binary solution.
- **Chapter IV-NATO Real Time Tactical Communications.** This chapter describes the operational details of LINK11 and LINK16 and the work NC3A has done to integrate XML into those tactical data links.
- **Chapter V-Experiments, Data Collection and Analysis.** This chapter compares network performance including compression and latency of XSBC and Fast Infoset.
- **Chapter VI-Conclusions and Future Work.** This chapter provides the recommendations based on completed work and suggestions for areas of further study.

THIS PAGE INTENTIONALLY LEFT BLANK

II. RELATED WORK AND BACKGROUND

A. INTRODUCTION

This chapter provides an overview to XML, Dial-a-Behavior protocol, XFSP, and Fast Infoset. Each of these technologies is similar in nature to or supported the development of XSBC.

B. XML

1. XML Characteristics

The Extensible Markup Language (XML) is useful for modeling data of all types. It can model very complex multi-layered data structures or something as simple as the roster for a baseball team as shown in Figure 1. It can be created and altered with any text editor. It is a way to markup data so that it can easily be understood by a computer.

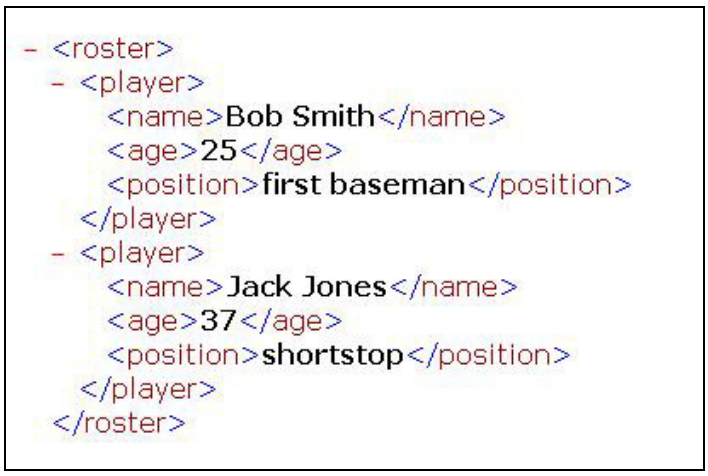

Figure 1. XML example showing self-describing tags and the ability to mark-up any type of data

XML has many similarities to the ubiquitous Hypertext Markup Language (HTML) that supports the bulk of the Internet. Referring back to Figure 1, one can see that it contains tags—words bracketed by '<' and '>'—to describe the data between them (Boss, 2003). This data used to describe other data is known as metadata. However, unlike HTML, all XML tags are defined by the user and do not have reserved or set meaning. While it is conventional to use tags that are helpful in describing the data, this

is not required. The first set of tags could be any combination of letters; it doesn't actually have to be a word, but well designed XML does have helpful element names and can be read and understood. It is in a text format only to offer ease of use in creating, editing, and debugging for developers. XML is designed to be intentionally verbose to facilitate ease of use. While text is almost always more expensive in terms of memory to store, compression algorithms have become more optimized and the cost of memory has decreased (Boss, 2003). This was a justification for the W3C to allow that property into the XML 1.0 specification.

It is also important to realize that XML is not an isolated technology, but the foundation for multiple technologies (Boss, 2003). These other developments include Extensible Style Language (XSL), Document Object Model (DOM), and XML Schema. XSL is a language based on XML used for expressing style sheets. DOM is a way of representing XML in memory and XML schema is a way to validate the structural integrity of a XML file.

XML is a relatively new technology, but has its roots in Standard Generalized Markup Language (SGML) as does HTML (Hunter et al., 2003). SGML was created to structure large complex documents and as a result is a very complicated yet powerful language. But SGML was too unwieldy for simpler applications and XML and HTML were born. There were designed to be subsets of SGML to be used for specific purposes and have served their purposes well. In keeping with language evolution, the W3C recently established a hybrid language of XML and HTML known as XHTML (Boss, 2003). It contains some of the syntactical structure of XML and is meant as a complete replacement for HTML.

One of the strengths of XML is its modularity. XML is used all over the world to model data and identical tag names are being used for different purposes. The tag <player> used on the baseball roster has a different meaning from the same tag used in selecting what media application to play an audio file. However, the concept of namespaces has allowed vocabularies of element names to be uniquely identified to avoid confusion. This allows for developments such as the Resources Description Framework which provides a common definition for various data items. Once

applications agree on the meaning of data, the data can be passed from application to application to further integrate functionality. An example is being able to identify people in a picture on the web from your e-mail address book (Boss, 2003). The picture and web browser both need to have the metadata available on the picture and be able to communicate to the e-mail application. The other strength of XML is that it is license free, platform independent, and well supported (Boss, 2003). The absence of a license and ability to operate on different operating systems promotes a broad base of support. License free means one can read, write, and develop in XML without having to pay for it as well as develop applications that use it. As a result, the open source community has produced a number of high quality and effective tools for XML free of charge. Internet searches for XML return millions of hits showing the expanse of the interest and support available.

2. XML Schema

XML Schema is an associate technology to XML. A schema document provides a definition of the structure and contents of a specific XML file (Hunter et al., 2003). It delineates the exact order of the elements found in a file, the name and type of the elements, and the name and type of attributes. Schema documents are themselves created using XML and a schema can be written for any XML file. Once the schema is created, only a small addition is needed in the XML file to reference the namespace of the schema. The two documents are now linked and the XML file can be tested to see if it conforms, or validates, to the schema. Multiple implementations of schema validation programs exist, but all basically function the same way. If a name of an element is misspelled or the elements are out of order, or if data is of the wrong type, then the documents will not validate but rather return a schema error. This verifies the structure and data types within the document.

3. XML Security

a. XML Canonicalization

The word *canonical* means a standard or a measurement for all others to compare against. While a standard already exists for well formed XML, flexibility still exists within those guidelines. Two documents must contain identical data, but one document might use empty elements while the other uses start and end tag pairs. Other differences include having elements in a different order and different handling of whitespace inside tags (Leung, 2004). These are just a few examples of how the same data might be represented in various seemingly insignificant variations.

Canonicalization is applying a rule set to resolve these differences into a standard form (Leung, 2004). Having a standard for how the XML needs to be consistently formatted is crucial when attempting to verify that two documents are identical. Digest algorithms, like MD5, create different hash values for two XML documents that contain the same Infoset but also contain differences allowed by the "well-formed" rule set. As a result, documents are canonicalized before determining their hash values. By canonicalizing XML files, one solves the problem with digest algorithms. For a complete specification of the canonicalized form consult the W3C's Canonical XML Version 1.0 document.

b. XML Signature

XML Signature is a method for incorporating proven public key encryption techniques into XML documents. The first step is for the signer to hash the document. This produces a string of characters or numbers. This string is then encrypted with the signer's private key and the resulting encrypted hash value is attached to the document. Upon receipt, the receiver uses the public key to decrypt the hash and compare it with a hash he has performed on the document. If the two values are identical then it is known that the document was not altered in any way (Leung, 2004).

The XML Signature standard can be utilized as an enveloped signature or a detached signature. An enveloped signature is alongside the data it is signing, inside the document, while a detached signature is signing data that is not part of XML

document containing the signature. In either implementation, none of the data in the signed document is actually encrypted. The hash value incorporated in the signature is only used to verify the integrity of the document, not encrypt the actual data. A signed document is still human readable.

Both uses of the XML Signature Recommendation employ the <Signature> element as defined by the standard W3C namespace (Leung, 2004). The <Signature> element contains two standard elements: <SignedInfo> and <Signature Value> (Leung, 2004). The former element contains information concerning the canonicalization algorithm to use in addition to the signature method used to sign the data. The <SignedInfo> element also contains references to data that is signed. The <Signature Value> element contains the actual hash or signed value. An optional element inside <Signature Value> is <KeyInfo> which contains information on how to retrieve the key or actually provides the key itself (Leung, 2004).

c. XML Encryption

XML Encryption differs from XML Signatures in that it replaces the data to be protected with <EncryptedData> element (Leung, 2004). This renders the encrypted portions of the data unreadable without the key. XML Encryption can be used to encrypt an entire document, particular elements or only the data within an element. The <EncryptedData> is much simpler than the <Signature> element. It contains an attribute to specifying the algorithm used and an element named <CipherData> which either encapsulates the actual cipher value or a reference on how to obtain it (Leung, 2004).

C. EVOLUTION OF A BINARY XML ENCODING

1. Dial-a-Behavior Protocol

Dial-a-Behavior Protocol was developed by NPS in order to dynamically change protocol syntax over DIS (McGregor, 2000). The Dial-a-Behavior protocol is a network solution for using multiple data formats or protocols without having to hard code each unique format. The common approach for UDP communication across a network, due

to bandwidth constraints, is to send data in binary form as a stream of bytes. While very efficient for transmission, this also requires the client to know the exact format of the data being sent so that it can parse the stream at the correct offsets to gather the data and then assign the correct units. To solve this issue, an XML file is created to describe the syntax of each format. These XML files are distributed to the clients and they now have a precise format to both write packets and receive packets. Additionally, any new protocols added to the network only require the addition of another XML file to describe the format.

2. Cross Format Schema Protocol (XFSP)

The cross format schema protocol (XFSP) was a solution for implementing a run-time extensible application layer protocol for a distributed system namely NPSNET-V (Serin, 2003). Ekrim Serin, a Turkish Naval Officer, sought to add flexibility to the NPSNET-V simulation by allowing for communication protocols to be changed or even created during run time. Traditionally, a programmer had to hard code the protocols into every host and then compile them which meant the entire simulation had to be brought to a halt and restarted after the changes were implemented. Due to the distributed nature of the network, homogeneity across the simulation hosts is practically impossible to achieve. Therefore, possessing the ability to tailor the communication protocols based on network performance or limitations of certain users provides an invaluable tool to those running the simulation.

Serin's research led him to Extensible Markup Language (XML). More precisely, it led him to the use of XML Schema, which provides a rigid structure for defining XML documents as an encoding basis for diverse documents. In this way, an XML schema might be used to define new protocols while the XML documents themselves were actually the transmitted messages. It is important to note that XFSP is a solution for creating a dynamic protocol and not the protocol semantics (which is a much more difficult problem). However, the verbose nature of XML does not lend itself well to be transmitted over a network with low bandwidth. Serin reasoned that if each user had the schema used to define the protocol, a simple substitution scheme could be implemented to reduce the physical size of the messages to be transmitted (Serin,

2003). Thus, he developed the Extensible Cross Format Schema Protocol. He then conducted generation and serialization trials to gather data to show the parsing time improvements available through XFSP.

XFSP also found use with Extensible 3D (X3D) graphics standard, a three dimensional modeling language that includes XML encodings. Modeling any object with high fidelity in three dimensions requires a considerable amount of data as information about thousands of data points in space have to be encoded. X3D is no exception which is why it proved to be a good candidate for XFSP. Using the GZIP compression algorithm in addition to using XFSP proved to create a file 78% smaller than the original (Serin, 2003).

D. FAST INFOSET

1. Overview

Fast Infoset is a component of Sun Microsystems Fast Web Services technology. An open source implementation of Fast Infoset was newly released in the Java Web Services Developer Pack 1.6. This is a solution for securing the benefits of networking via XML without having to suffer the costs of parsing and serializing text XML documents. The uses of XML in various applications are multiplying rapidly as the technology matures, but even a cursory review of XML reveals many advantages. Primarily XML enables cross-platform data exchange as any XML parser can extract the information from the document. Also, with user-defined tags, developers are free to structure the data in the most useful way without being constrained to specified tags. However, the cost in time and CPU cycles to parse and serialize XML documents can be considerable due to the redundant and verbose nature of XML. Many XML documents occupy more memory and are more computationally expensive than their binary equivalents. This makes its use difficult by mobile devices that are limited by battery life and heavily constrained by processing power. Fast Infoset seeks to strike balance by reducing both XML file sizes and processing time at the expense of human readability (Sandoz & Percias-Geersten, 2005). It is a single open-source binary XML solution among many that are seeking to accomplish the same tasks. However, Sun

has submitted the ITU-T Rec. X.891 | ISO/IEC 24824-1 (Fast Infoset) specification to both ITU-T and ISO for standardization (Sandoz & Percias-Geersten, 2005).

2. Algorithm Description

As the name implies, Fast Infoset relies upon the W3C XML Information Set, a layer of XML abstraction, in order to determine the structure of the document for compression purposes (Cowan & Tobin, 2004). More specifically, the XML Information Set is a collection of standardized definitions that can be used to map out the structure of any XML document (Gudgin, 2004). It can be thought of as a conceptual representation of a DOM tree. The tree structure and its nodes are referred to as the information set and information items, respectively. Utilizing the tree concept, Fast Infoset maintains the hierarchical structure of the XML document. Taking advantage of this standardized structure Fast Infoset is able to implement its compression algorithm in lieu of using an XML Schema. However, a schema can be used for validation to verify the round tripability of the document. This independence from a schema adds flexibility to this particular binary XML implementation (Sandoz, Triglia, & Percias-Geertsen, 2004).

Figure 2 shows the simple flow of how an XML document is first serialized to produce a fast Infoset value which in turn is encoded to produce the fast Infoset documents. It is this document that is actually submitted for transportation. While similar to a DOM tree, the XML Infoset can be streamed and does not have to be stored entirely in memory in order to serialize it. The Fast Infoset value that is created in the intermediate step is then encoded by the compression scheme.

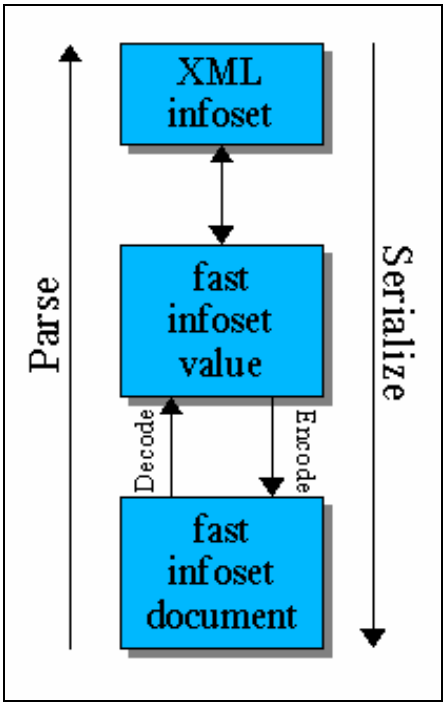

Figure 2. Fast Infoset relationship diagram showing that an XML Infoset is serialized into a Fast Infoset value and then document. Conversely, parsing takes a Fast Infoset document and produces a Fast Infoset value and finally a normal XML Infoset (Sandoz et al., 2004)

The principal method used for compression is indexing recurring strings (Sandoz et al., 2004). As the document is serialized each common string is noted and placed into a table. Any additional occurrence of the string is replaced with an index that references the appropriate string in the table. During deserialization, the parser creates an identical table by indexing the first occurrence, and then replaces the index with the string value found in the table. Another compression method applies the same logic to qualified names. The qualified names are indexed which when decoded are referenced to an actual object which eliminates the need for redundant calculations to determine the namespace of a particular element information item or attribute information item (Sandoz et al., 2004).

Efforts were also made to streamline the serializing and parsing processes. An example is how end tags are encoded. While start tags are indexed as described above, end tags are handled differently. The Fast Infoset document does not require

end tags because it uses the start tags to automatically generate appropriate end tags during the parsing process. As a result, all end tags are simply marked with a reserved integer value so they can be identified during parsing. In addition, Fast Infoset does not check for escaping of character data in an effort to streamline the serialization process (Sandoz et al., 2004).

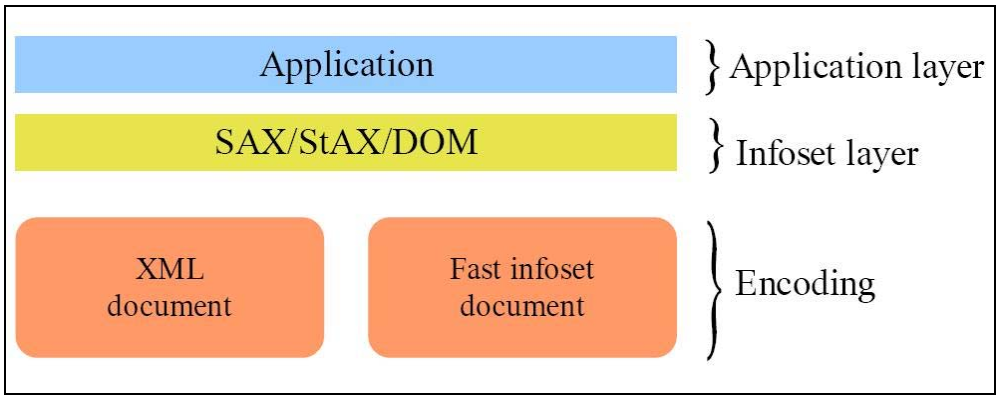

Figure 3. Placement of the Fast Infoset document in the XML stack. A Fast Infoset document is an alternate encoding to XML (Sandoz, 2005)

Fast Infoset is intended to be completely integrated into current systems using XML. It seeks to be completely transparent to the user by becoming an alternate encoding and thus finds its place in the XML stack as shown in Figure 3.

An API is provided for further development, but most users will enjoy the benefits of it without being aware of an additional software module running before transmission. Fast Infoset produces XML documents that are completely round-trippable as long as a DTD is not referenced within the document (Sandoz & Percias-Geersten, 2005). The reason for this is that Fast Infoset was designed to operate independently of DTDs and cannot perform the necessary dynamic computations to determine properties from the minimalist information provided in a DTD (Sandoz & Percias-Geersten, 2005). With this independence from a schema or DTD the actual XML document need only to be serialized into the Infoset layer on the stack via SAX, StAX, or another implementation to be round-trippable. This shows the motivation for Sun to seek standardization as this format might become a popular encoding alternative for the family of technologies based on XML

E. EXTENSIBLE 3D (X3D) COMPRESSED BINARY ENCODING (ECB)

X3D is an XML format for describing 3D graphic objects for communication applications. It provides many improvements over its processor, the Virtual Reality Modeling Language (VRML) (Web3D Consortium, 2005a). By representing the 3D data in XML, X3D also provides real-time extensibility as well as easy integration into web services. XML also allows the 3D content to be used on various platforms.

1. X3D CBE Requirements

In 2003 the Web3D Consortium released a request for proposals for an X3D Compressed Binary Encoding (Web3D Consortium, 2003). This request was made to address the memory issues involved in reducing X3D file size over a network. XML based networking was of particular concern due to Web3D's strategies for web services and real-time applications. As a result, the X3D working group created the following requirements for the compressed binary encoding. Figure 4 shows the results of the working group. Note the requirements under "Multiple, Separable Data Types" are the specific data types desired to be supported.

1. **X3D Compatibility**. The compressed binary encoding shall be able to encode all of the abstract functionality described in X3D Abstract Specification.
2. **Interoperability**. The compressed binary encoding shall contain identical information to the other X3D encodings (XML and Classic VRML). It shall support an identical round-trip conversion between the X3D encodings.
3. **Multiple, separable data types.** The compressed binary encoding shall support multiple, separable media data types, including all node (element) and field (attribute) types in X3D. In particular, it shall include geometric compression for the following.
    - **Geometry** - polygons and surfaces, including NURBS
    - **Interpolation data** - spline and animation data, including particularly long sequences such as motion capture (also see Streaming requirement)
    - **Textures** - PixelTexture, other texture and multitexture formats (also see Bundling requirement)
    - **Array Datatypes** - arrays of generic and geometric data types
    - **Tokens** - tags, element and attribute descriptors, or field and node textual headers
4. **Processing Performance**. The compressed binary encoding shall be easy and efficient to process in a runtime environment. Outputs must include directly typed scene-graph data structures, not just strings which might then need another parsing pass. End-to-end processing performance for construction of a scene-graph as in-memory typed data structures (i.e. decompression and deserialization) shall be superior to that offered by gzip and string parsing.
5. **Ease of Implementation**. Binary compression algorithms shall be easy to implement, as demonstrated by the ongoing Web3D requirement for multiple implementations. Two (or more) implementations are needed for eventual advancement, including at least one open-source implementation.
6. **Streaming**. Compressed binary encoding will operate in a variety of network-streaming environments, including http and sockets, at various (high and low) bandwidths. Local file retrieval of such files shall remain feasible and practical.
7. **Authorability**. Compressed binary encoding shall consist of implementable compression and decompression algorithms that may be used during scene-authoring preparation, network delivery and run-time viewing.
8. **Compression**. Compressed binary encoding algorithms will together enable effective compression of diverse datatypes. At a minimum, such algorithms shall support lossless compression. Lossy compression alternatives may also be supported. When compression results are claimed by proposal submitters, both lossless and lossy characteristics must be described and quantified.
9. **Security**. Compressed binary encoding will optionally enable security, content protection, privacy preferences and metadata such as encryption, conditional access, and watermarking. Default solutions are those defined by the W3C Recommendations for XML Encryption and XML Signature.
10. **Bundling**. Mechanisms for bundling multiple files (e.g. X3D scene, Inlined subscenes, image files, audio file, etc.) into a single archive file will be considered.
11. **Intellectual Property Rights (IPR)**. All technology submissions must follow the predeclaration requirements of the Web3D Consortium IPR policy in order to be considered for inclusion.

Figure 4. Requirements of a binary encoding for X3D to include sub-categories of supported data types (Web3D Consortium, 2003)

Most of the requirements are straightforward. X3D compatibility requires the encoding to be able to represent all the functionality of X3D. Interoperability requires the coding to be able to round trip through XML, VRML, and normal X3D (Web3D Consortium, 2005a). This ensures backwards compatibility. The streaming requirement is logical as X3D is meant for communication with interactive applications which places a heavy emphasis on responsiveness. Requirements of interest are the security and bundling requirements. The security requirement specifies support for the XML Encryption and XML Signature standards to be able to protect 3D content (Web3D Consortium, 2005a). In addition, access management and privacy controls were desired. The bundling property explores the ability of the encoding to allow X3D to be self contained by consolidating X3D scenes, images, and audio files into a single file.

2. Interim Solution

The Web3D Consortium evaluated generic binary XML compression candidates. Once the specific solution was chosen continued work on performance tuning and optimization can now be continued. XSBC was considered by the Consortium along with Sun's Fast Infoset implementation. After a careful review of performance standards and requirements, it was decided that Fast Infoset provided the best specified capabilities and thus chosen as the solution. It was then included in the draft ISO/IEC 19776 standard for binary encodings of X3D (Web3D Consortium, 2005b). If a W3C standard emerges for binary XML, that recommendation will supersede Fast Infoset as the bases for information-theoretic compression.

F. SUMMARY

This chapter describes the structure and functionality of XML and its related technologies in addition to introducing and describing XFSP and Fast Infoset which are two binary XML encodings.

THIS PAGE INTENTIONALLY LEFT BLANK

III. XML SCHEMA-BASED BINARY COMPRESSION (XSBC)

A. INTRODUCTION

This chapter gives a brief history of XSBC and describes the serialization and deserialization processes and current applications of XSBC.

B. BACKGROUND

Cross Format Schema Protocol (XFSP) was developed by Ekrim Serin, a computer science student at NPS, for the transmission of schema based networking protocols. He wanted to show that large networked virtual environments could reliably run entirely on dynamic schema protocols (Serin, 2003). XFSP provided a way to take protocols defined in XML, serialize them for transmission over the network, and then deserialize and be used by entities in the simulation. His source code was enhanced by Yumetech Inc. using modern software engineering practices and design patterns. NPS then changed the name of the project to XML Schema based Binary Compression (XSBC).

C. XML SERIALIZATION

The first step in the algorithm is to parse the schema associated with the file to be serialized. As the schema is parsed XSBC generates tables to hold information that can be referenced later during the tokenization. An element table records the XPath expression of an element as well as its datatype. In addition, unique integer values (tokens) are assigned to both the start and end tags. For attributes, a separate table is constructing containing the XPath and datatype as well as a token to represent the attribute name. The integer values 0, 1, and 2 are not assigned to any element or attribute but are used to delineate attributes, identify elements that contain data, and other special cases (Serin, 2003). Once these tables are created, the XPath and token values are used to create hash tables for quicker reference (Serin, 2003).

Problems arise during schema parsing when attributes or elements have identical names. XSBC needs to ensure that unique token values are assigned to each

attribute and element for correct deserialization. Therefore, XSBC examines parent element names and data types in order to make these distinctions.

Once the tables have been created XSBC replaces all element tags with the integer tokens stored in the hash table. The same replacement is also preformed for attribute names. It is important to note that none of the payload data is indexed and replaced with an integer, but is serialized into its appropriate binary representation. Figure 5 provides a basic representation of this process with the caveat that the integer token replaces the tag brackets “<” and “>” in addition to the text inside. The brackets remain in the figure to facilitate recognition of the replacement process. The top section shows the original XML file with its descriptive tags while the bottom displays a representation of the file after the replacement algorithm has run. This replacement process and the binary payload conversion are what are responsible for the reduced XSBC file size. The memory required to store a element name is one byte for each character in the string plus two more bytes—one for each of the tag brackets “<” and “>”. A short integer however requires only 2 bytes (Sun Microsystems, 2005). Since the token requires the same amount of memory as the brackets, the token will always use less memory than entire element or attribute name.

```
<?xml version="1.0" encoding="UTF-8"?>
<protocol xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="example.xsd">
    <location x="3.45" y="56.72" z="-10.1"/>
    <header>
        <exerciseID>1</exerciseID>
        <version>1</version>
        <pdutype>2</pdutype>
    </header>
    <velocity x="1.0" y="0.0" z="-0.7"/>
</protocol>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<10 >
    <14 24="3.45" 25="56.72" 26="-10.1" 15>
    <12>
        <20>1<21>
        <18>1<19>
        <22>2<23>
    <13>
    <16 27="1.0" 28="0.0" 29="-0.7" 17>
<11>
```

Figure 5. The top pane shows normal XML document. The bottom pane shows the same document after XSBC has replaced element and attribute names with integer tokens (Serin, 2003)

In addition, XSBC also seeks to automatically perform useful type conversions. If the schema indicates a marked-up string value is a number, then XSBC will convert it into a binary numeric representation so that it is recognized not as text but as a number. This is useful for data binding purposes such as assigning the value to Java variable.

XSBC also seeks to reduce file size by examining the type of the data payload to see if it can be simplified to a representation that occupies fewer bytes. An attribute of type long, for example, will allocate eight bytes of memory (Sun Microsystems, 2005). If XSBC determines that no data loss will occur, it will convert the data to an integer or a short integer to save space. While this process adds additional processing time, it also produces more efficient compression. Once all replacements and type conversions are complete the document is serialized, via DOM4J, and streamed over the network.

D. XML DESERIALIZATION

Deserialization is a straightforward process and it begins by receiving the data stream and capturing the tag numbers. These tag numbers are then compared to the table created during serialization in order to retrieve the element or attribute associated with it. After the tags are resolved, the data is read from the stream in its binary form (Serin, 2003). Once the tags are used to create elements and attributes, the data is then bound to them. As the tags and data arrive, the order and structure of the XML document is rebuilt through a series of simple stack operations. The reconstruction of an example XML file is shown in Figure 6. Once this process is complete a schema can be used to validate the integrity of the newly rebuilt file.

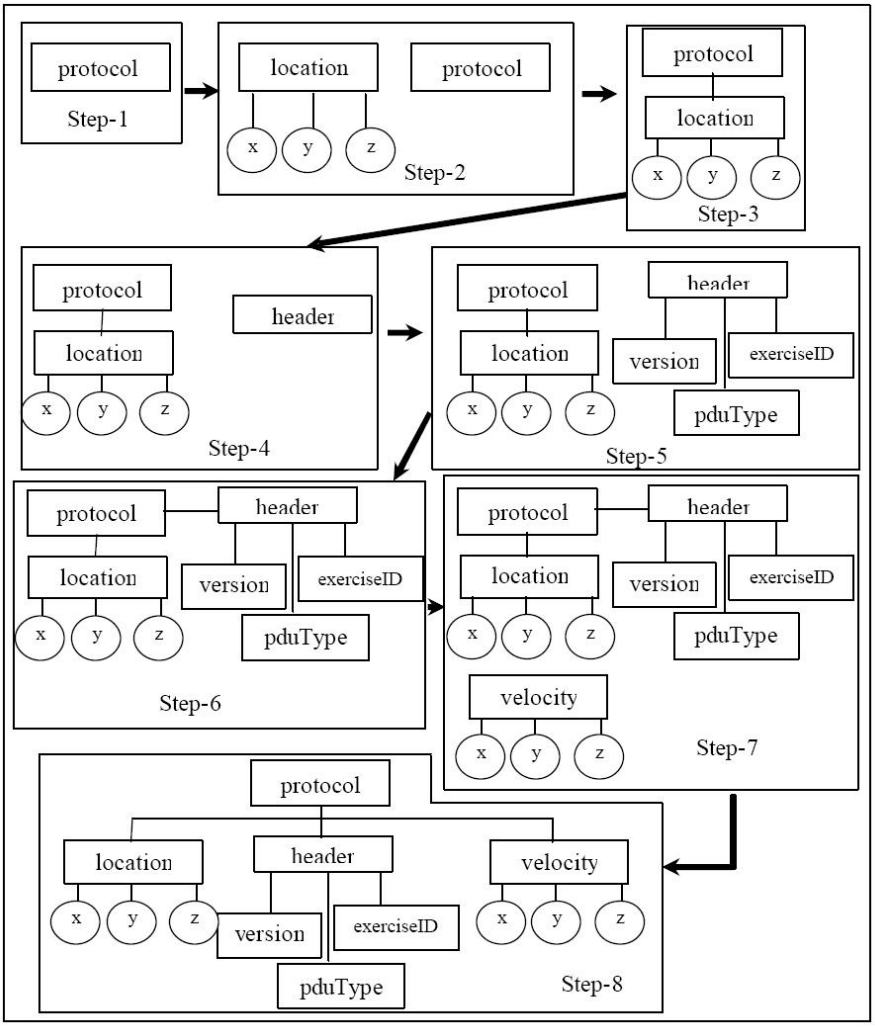

Figure 6. Sample XSBC deserialization from a stream and XML tree reconstruction (Serin, 2003)

E. CURRENT USES

XSBC has a myriad of potential applications since it can operate on any well-formed XML document defined by a schema. However, the compression benefits tend to make it more readily applicable to areas where bandwidth is a significant concern. It also can aid applications which are required to parse large XML files, as the binary encoding simplifies this process. Below are current applications that are utilizing XSBC.

1. AUV Workbench

The Autonomous Unmanned Vehicle Workbench is an ongoing work at NPS to develop a robust toolkit for AUV simulations. The workbench is built entirely from Java, XML, and X3D. It is used for AUV mission planning and will eventually be used to execute commands to control the robot.

The Workbench accurately models an underwater environment, and can also be used to evaluate sonar modeling and hydrodynamics responses for future AUVs. A centrally important feature is the 3D visualization of the model while executing the mission provided via X3D. One can also load an old archived mission file and view the playback for mission analysis.

This playback capability is available because as the AUV model executes its mission it streams all the mission results via TCP sockets. The mission results file, encoded in XML, contains all the commands executed by the AUV as well as position data. The generated TCP stream is captured by an XSBC server thread which subsequently transforms the data in a binary XML format for archiving purposes. Currently, this TCP stream is a loopback, to demonstrate this capability, but MOVES has conducted experiments in which the AUV robot, within the simulation, was remotely controlled via a VPN streamed the data to the remote location. One goal of the Workbench is to actually control the physical AUV robot via XML command sent acoustically using XSBC. The robot can then stream data back to the surface in the same manner. In order to support such a difficult transmission medium forward error correction (FEC) has also been added to the XSBC module with in the AUV Workbench (Norbraten, 2004).

2. XJ3D

XJ3D is a set of open source tools for handling VRML97 and X3D content. Written completely in Java and open source, it is a portable and free solution for rendering X3D images. The X3DElementReader class implements XSBC's ElementReader interface and uses it to parse and handle data. The ElementReader class, while similar in functionality to the SAX ContentHandler, does not convert the data into a string representation but leaves it in its binary form. This prevents any unnecessary conversions to a string by keeping the data in a binary format.

3. XSBC Comparison Tool

The XSBC Comparison Tool is a GUI interface for XSBC encoding that captures and displays the parsing and compression performance in regard to a particular file chosen by the user. The ultimate goal for this tool is to output the compression statistics into an XML tagset which can be used as an exemplar regression test, but currently it is an excellent way to discover the exact performance benefits XSBC has for particular XML files.

GNU Zip or GZIP is also incorporated by the comparison tool to provide additional compression after the XSBC encoding. After processing, the selected XML file the tool displays following metrics: the original file size, the file size with GZIP applied, the XSBC encoded file size, and the XSBC encoded file with GZIP applied. In addition the tool also captures and displays machine independent parsing times for each of the compression configurations. A sample file was processed and is displayed in Figure 7. This tool is bundled with the XSBC code and is open source and available on the Internet. (See Appendix C) It is important to note that since XSBC is reliant upon a schema for encoding, the XSD file for the desired XML must also be co-located with the selected XML file to process.

The tool has also been further modified to include the same comparison support for Fast Infoset. A more detailed analysis of both XSBC and Fast Infoset performance is located in chapter V in which this tool was used extensively.

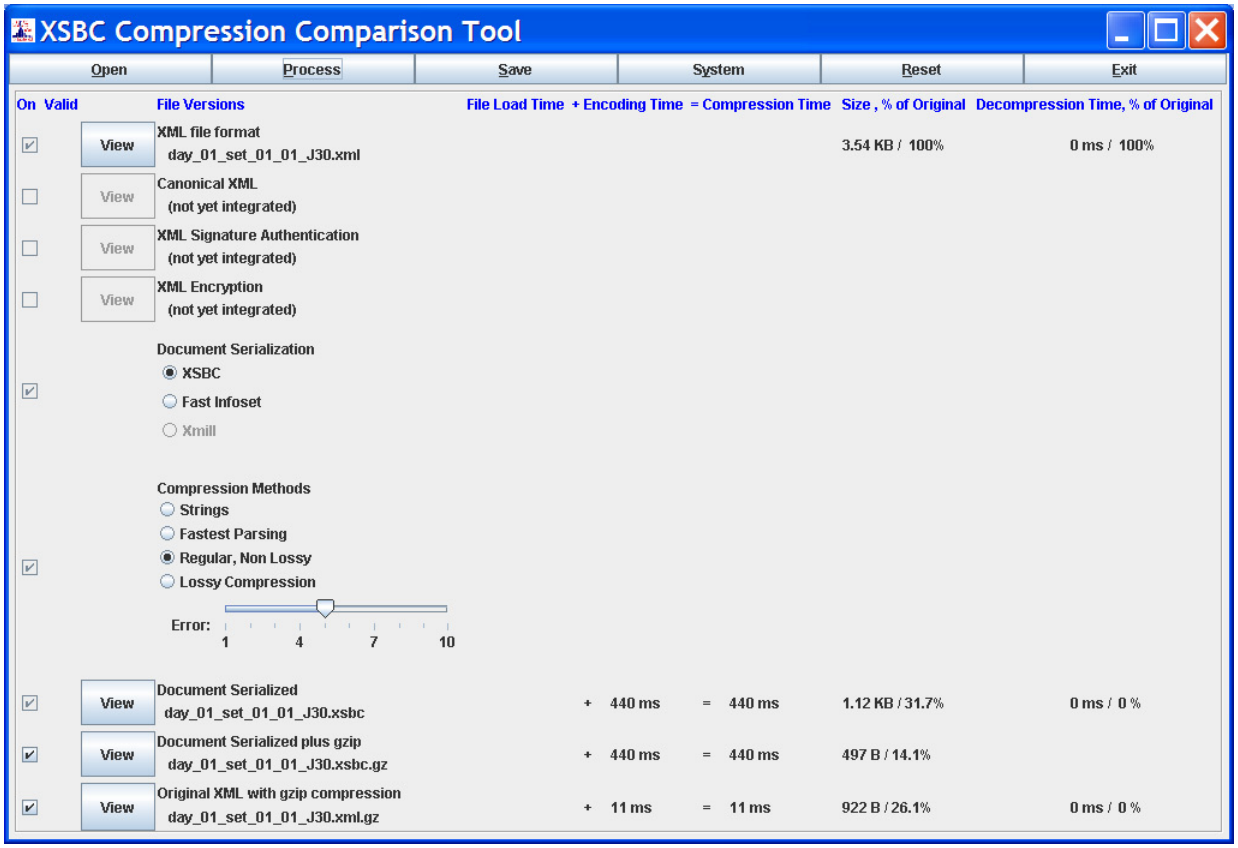

Figure 7. XSBC Comparison Tool showing a 3.54KB File being compressed by 85.9%

F. POTENTIAL USES

As previously mentioned, XSBC was originally developed as a way to create dynamic network protocols for virtual environments. This allows for applications to handle various packet payloads that have been created on the fly. These protocols, based on XSBC, can be implemented on any network and the protocols themselves customized to meet specific needs.

As demonstrated by the AUV Workbench, XSBC can be used in conjunction with streaming data and is not restricted to operating on static files. This opens the door for web applications. Other real time streaming applications are also possible such as rendering 3D scenes.

G. SUMMARY

This chapter described the algorithm by which XSBC serializes and deserializes XML documents. It also shows some current applications of XSBC.

IV. EVOLUTION OF BINARY XML

A. INTRODUCTION

This chapter follows the process by which the W3C determined the requirements to be met by binary XML, as well as the properties a standard encoding should have and how to proceed in the future. It also examines XSBC against the properties a proposed standard must contain.

B. W3C BINARY WORKSHOP

In September of 2003, W3C organized and conducted the W3C Workshop on Binary Interchange of XML Information Item Sets (Lilley & Karmarkar, 2003). The workshop contained an array of international members including professionals from multiple software vendors, various academic institutions and a US defense think tank (Lilley & Karmarkar, 2003). The end goal for the workshop was to decide whether or not the W3C should charter a working group for further investigation into adopting a standard for a binary XML encoding (Lilley et al., 2003). The three day conference consisted of various members presenting either their proprietary binary XML solution and/or concerns in regard to establishing an internationally recognized standard. The discussion ranged from discouraging the formation of a standard to optimistic specific use cases to be considered while testing candidate solutions.

1. Arguments against Binary XML

Those in opposition to the creation of a standard fall into three categories: those that think a viable binary XML encoding is not currently feasible, those that believe it is not necessary, and those opposed to the costs of deploying a new format.

The members opposed to a binary standard cited the multiple and varied uses of XML which produce equally varied requirements and priorities in optimization. The properties developers are most interested in trying to optimize include memory footprint, parsing time, and generation time.

Mobile devices are constrained in physical size which also greatly limits the amount of physical memory available. Also, wireless networks lag behind wired networks in bandwidth availability. As a result, in order for these devices to take full advantage of XML, an efficient way to transmit and store these documents is needed. In this scenario, memory footprint of the binary encoding takes priority over generation and parsing speed. However, mobile devices are additionally limited in processing power and, more critically, battery life. As a result, processing efficiency is a significant benefit in mobile applications.

On the other end of the spectrum is a high-performance server with memory to spare. In this instance, the document generation and parsing times for incoming documents is emphasized. Memory is plentiful and upgradeable, but time cannot be regained. These examples exhibit the existing tradeoff between a smaller memory footprint and longer parse time as it is extremely difficult to increase the compression of a file format without incurring any additional processing costs (Pal, Marsh, & Layman, 2003). This may suggest that, in lieu of a single standard, multiple formats might be utilized, but this is counter-productive to the overall goals of XML considering the motivations for establishing a standard in the first place (Pal et al., 2003). Creating multiple standards that are each optimized for a specific use case is detrimental to keeping XML interoperable which has proven to be one of its greatest strengths. This leads one to the conclusion that a single standard, if it satisfies the requirements, is greatly preferred. However, some argue that a single standard might be too compromise-saturated so that it provides universally mediocre performance which may not meet the needs of any particular use case that prompted any compromise (Pal et al., 2003).

Others argue that not enough evidence exists to merit a new encoding; that the majority of the requirements used to support the need for binary XML can be met through optimizing XML or one of its familial technologies (Conner & Mendelsohn, 2003). The idea is that developments (such as cheap/fast memory and processing) in current technologies can solve many of these issues. Coupled with this thought is the idea that Moore's law, which states that processing power roughly doubles every 18 months, has enabled developers to stray from the discipline of striving to code the

smallest, most efficient solutions (Wikipedia, 2005). Another concern, in a similar line of thought, is whether any useful encoding developed now will not become obsolete as processing power increases rapidly according to Moore's law. The encoding must consist of improvements of a substantial nature such that it will continue to be valuable beyond just the near future.

2. Arguments for Binary XML

The consensus of the participating working group members, however, simply disagreed with the dissenting minority's previous evaluation of both the state of XML and the need for an alternate binary encoding (Lilley et al., 2003). The primary view of these members was that a real need for an additional encoding exists and, although a perfect solution does not, a feasible solution answering a sufficient set of requirements can be found.

The sheer number of fledgling solutions appearing in the market provides evidence enough that a real need exits and deserves advanced optimal solutions. The Workshop was encouraged to approach the issue as an engineering problem and choose a solution based on the 80/20 principle (Orchard, 2003). This means choosing a solution that offers the greatest gain in performance, while losing the least number of other properties, and satisfies the greatest proportion of the use cases. It may be true that a selected standard might not be the optimal solution for particular instances, but the benefit of widespread acceptance and use can greatly outweigh those outlying scenarios. This approach was successfully undertaken when XML itself was introduced.

The crucial elements to consider in the 80/20 engineering process are the use cases. One must be careful to discern between the needs for greater XML performance that are attributed to the design of XML and which are the result of inefficient applications, verbose data, etc. (Orchard, 2003). The scenarios that require a novel solution which will not be fixed by faster processors or more memory are the scenarios that become the use cases which, later on, prove to be invaluable. These use cases help to prioritize technical requirements.

At the close of the workshop, which entailed three days of discussion, an informed consensus agreement was reached among the members present that the W3C should further investigate Binary XML and charter a working group as an appropriate forum to do so (Lilley et al., 2003).

C. XML BINARY CHARACTERIZATION (XBC) WORKING GROUP

1. Introduction

In May of 2004 the XML Binary Characterization Working Group conducted its first face to face meeting. The charter for the working group specified that the following deliverables were to be produced by March of 2005: a use cases document, a properties document, and a characterization document. The documents are complete, publicly available via the W3C website, and are reviewed below.

2. XML Binary Use Cases

Use cases are simply realistic scenarios where XML, due to its design, has proven to fall short in providing an adequate solution. Care was taken in to ensure that the final eighteen approved use cases covered all aspects of XML utilization. Below are all eighteen use cases and descriptions of the four use cases that are relevant to this work (M. Cokus & Percias-Geertsen, 2005a).

W3C XML Binary Use Cases
1. Metadata in Broadcast Systems
2. Floating Point Arrays in the Energy Indu
3. X3D Graphics Model Compression, Serialization, and Transmission
4. Web Services for Small Devices
5. Web Services within the Enterprise
6. Electronic Documents
7. FIXML in the Securities Industry
8. Multimedia XML Documents for Mobile Handsets
9. Intra/Inter Business Communication
10. XMPP Instant Messaging Compression
11. XML Documents in Persistent Store
12. Business and Knowledge Processing
13. XML Content-based Routing and Publish Subscription
14. Web Services Routing
15. Military Information Interoperability
16. SyncML for Data Synchronization
17. Sensor Processing and Communication
18. Supercomputing and Grid Processing

Table 1. Use cases assembled by XBC for a binary XML encoding (M. Cokus & Percias-Geertsen, 2005b)

a. Web Services for Small Devices

In the past 10 years handheld devices have become more powerful and user friendly and thus gained popularity at a staggering pace. The line between a cell phones and Personal Digital Assistants (PDAs) is becoming blurry and these are not the only devices with considerable wireless capability. Web services of all types are becoming more widespread and handhelds are quickly taking advantage of the benefits. However, these devices are limited in processing power, available memory, and battery life. This use case is particularly focusing on devices that are limited to a code size of 64KB and a heap size of 230KB (M. Cokus & Percias-Geertsen, 2005a). These constraints are especially profound as XML, which lies at the core of web services, has a large memory footprint. Consequently, a faster serialization process that generates smaller packets for transmission would greatly expand possibilities in this realm. The quicker parsing and serialization process will help reduces battery consumption and the smaller packets will enhance limited help reduce network capacity usage.

b. Military Information Interoperability

The US Department of Defense and its allies have developed into enormous enterprises in which timely and accurate information exchange is absolutely crucial. This is difficult as the data that traverses the enterprise as well as the elements that create the data are extremely varied. Interoperability throughout the enterprise is paramount and many Commercial Off the Shelf (COTS) systems are being utilized in creating web services to meet that need (M. Cokus & Percias-Geertsen, 2005a). XML provides an excellent solution as it can be used to represent data that is loosely coupled and provides for backwards compatibility, but a binary encoding of XML is needed for processing and networking efficiency. Backwards compatibility is an issue due to the extensive testing that must be conducted on any new system, therefore, integration of new technology is achieved piece meal throughout the enterprise usually over a considerable amount of time (M. Cokus & Percias-Geertsen, 2005a). As a result, at any one time, multiple versions of systems and software exist and each must maintain interoperability with the other. Also, the military is not immune to the bandwidth issues found in other test cases and may be most affected. This makes the prospect of a binary XML solution all the more appealing for the military use case.

c. Sensor Processing and Communication

The need for more accurate intelligence data by both military and civilian agencies has caused an increase in sensor architectures design (M. Cokus & Percias-Geertsen, 2005a). These architectures range from isolated static sensors with limited resources to mobile sensor arrays that dynamically interconnect to create networks. Data flow also varies as some sensors may only relay a data feed, but many also receive command and control instructions based on processed data reports. XML could provide a solution for communication and reporting protocols since it enables connections between dissimilar systems and can easily be put into human readable forms for interpretation or editing. However, the networks that connect sensors are often unreliable or lacking in available bandwidth (M. Cokus & Percias-Geertsen, 2005a). The sensors themselves could be battery powered and static sensors require an extended duration of time to gather useful data. These concerns point towards a

binary XML encoding that could provide instructions to sensors and receive reports in the smallest packet size possible. This minimizes bandwidth which, in turn, conserves battery power for unattended systems. Finally, due to the security nature required from some sensors, the ability to encrypt and decrypt without any additional overhead would be a desired characteristic (M. Cokus & Percias-Geertsen, 2005a).

d. Extensible Messaging and Presence Protocol (XMPP) Instant Messaging Compression

The Jabber instant messaging service is an increasingly popular tool that is built on XMPP functionality. XMPP is a streaming XML protocol that provides instant messaging, group chat, and other real-time functionality (M. Cokus & Percias-Geertsen, 2005a). XMPP uses a simple client-server architecture in which XML fragments, known as stanzas, are exchanged to provide chat capability. A few of XMPP's advantages are flexibility and ease of use. The XML format allows additional protocols to be easily added as well as providing human readability for debugging (M. Cokus & Percias-Geertsen, 2005a). When stanzas arrive at the server for routing, they must be parsed in order to determine the ID of the client and the appropriate server to which the message needs to be forwarded. Even though the stanzas are small, high traffic scenarios with hundreds of thousands of stanzas to be parsed add up to create a large workload for XMPP servers (M. Cokus & Percias-Geertsen, 2005a). In addition, stanzas also add up to use a large amount of bandwidth. As a result, a binary encoding that reduces parsing times and serialized file sizes provides greater capacity and efficiency to XMPP.

3. XML Binary Properties

a. Introduction

After the use cases were established, they were examined extensively and the properties of binary encoding required by the use cases were recorded. Similar properties were combined and a final list was compiled. In this way the specific requirements for the encoding are established so that if the candidate solution contains all the properties listed in Table 2, it should theoretically work for all the use cases.

Algorithmic Properties	Generality
Processing Efficiency	Human Language Neutral
Small Footprint	Human Readable and Editable
Space Efficiency	Integratable into XML Stack
	Localized Changes
Format Properties	No Arbitrary Limits
Accelerated Sequential Access	Platform Neutrality
Compactness	Random Access
Content Type Management	Robustness
Deltas	Roundtrip Support
Directly Readable and Writable	Schema Extensions and Deviations
Efficient Update	Schema Instance Change Resilience
Embedding Support	Self Contained
Encryptable	Signable
Explicit Typing	Specialized codecs
Extension Points	Streamable
Format Version Identification	Support for Error Correction
Fragmentable	Transport Independence

Table 2. Desired algorithmic and format properties of a binary XML standard as derived from use cases by the W3C (M. Cokus & Percias-Geertsen, 2005b)

b. Additional Considerations

A few qualities desired of the binary encoding were agreed upon but not added to the properties list because of the difficulty in establishing an effective standard of measure by which to judge candidate solutions (M. Cokus & Percias-Geertsen, 2005b). These qualities include forward compatibility, low implementation cost, royalty free, and widespread adoption.

Forward compatibility is essential for the new format as data models are very dynamic. It is the idea that the implementation must flexible enough to be able to represent these new models through an augmentation of the standard (M. Cokus & Percias-Geertsen, 2005b). Forward compatibility is much more difficult and requires more careful planning than traditional backwards compatibility.

For the case of binary XML encoding, a natural solution to forward compatibility presents itself. Any future binary format can be supported by simply

translating intermediate formats back into XML. This approach as been adopted by XSBC and X3D, allowing current use of binary XML to proceed with confidence while work on future solutions continues. XML has displayed excellent forward compatibility as has changed very little while accompanying new data models.

Widespread adoption is another important consideration that is difficult to quantify. The new format should be able to provide solutions to a number of architectures and wide range of devices and not be pigeon-holed for a specific use. This is important as it creates an underlying support structure for the standard (M. Cokus & Percias-Geertsen, 2005b). Readily available support in turn aids in widespread adoption and the cycle repeats itself. When a new format begins to gain popularity various tools and resources begin to appear at an alarming rate, due in part, to the global nature of the Internet and the technical interest and savvy of many of its users. This helps provide a sense of legitimacy to the standard as it becomes well understood and supported.

Additional factors that encourage widespread adoption are a low implementation cost and a royalty free standard (M. Cokus & Percias-Geertsen, 2005b). Developing applications that implement the new binary format should be a relatively straightforward process. This entails a coding mechanism that does not require huge amounts of time as this increases cost by having to contract out to programmers. As XML already fulfills this requirement well, the code should seek to be XML compatible at the processing level or close to it. This approach enables the new format to be processed by current systems and thus lower the implementation cost. The last consideration, but one not to be overlooked, is a standard that is royalty free. This is a huge factor in encouraging widespread adoption as there is no cost for using the format or developing tools that process it. A royalty free standard can also be endorsed by the W3C which can bolster confidence in adopting the new standard (M. Cokus & Percias-Geertsen, 2005b).

4. XML Binary Characterization

a. Definition of Binary XML

In order to avoid ambiguity and clearly light the path to a standard, a solid definition must be constructed. Binary XML is defined as "a format which does not conform to the XML specification yet maintains a well-defined, useful relationship with XML." (Goldman & Lenkov, 2005) The W3C further defines useful to mean "that practical systems may take advantage of this relationship with little effort [such as] it may be use to convert a file from XML to binary XML." (Goldman & Lenkov, 2005) It is important to understand that binary XML is not simply a compression algorithm, but an entirely separate way to encode data that shares some of the same characteristics as XML but is totally independent from it. If a standard is adopted, systems will not need to convert from binary XML to pure XML in order to understand it, but will be able to understand and manipulate the binary encoding itself.

b. Development of Minimum Requirements

It was recognized by the working group that the initial list of 38 requirements and additional considerations was much too large. Trying to satisfy this considerably large number of requirements would certainly doom any prospective format to failure (Goldman & Lenkov, 2005). While all the requirements were important, it was agreed a measure of significance needed to be created to rank the requirements. The following categories were subsequently created: must have, should have, and nice to have. Each of the properties contained in every use case was sorted into one of three categories. From this a list of must have properties was obtained. This process only eliminated seven properties.

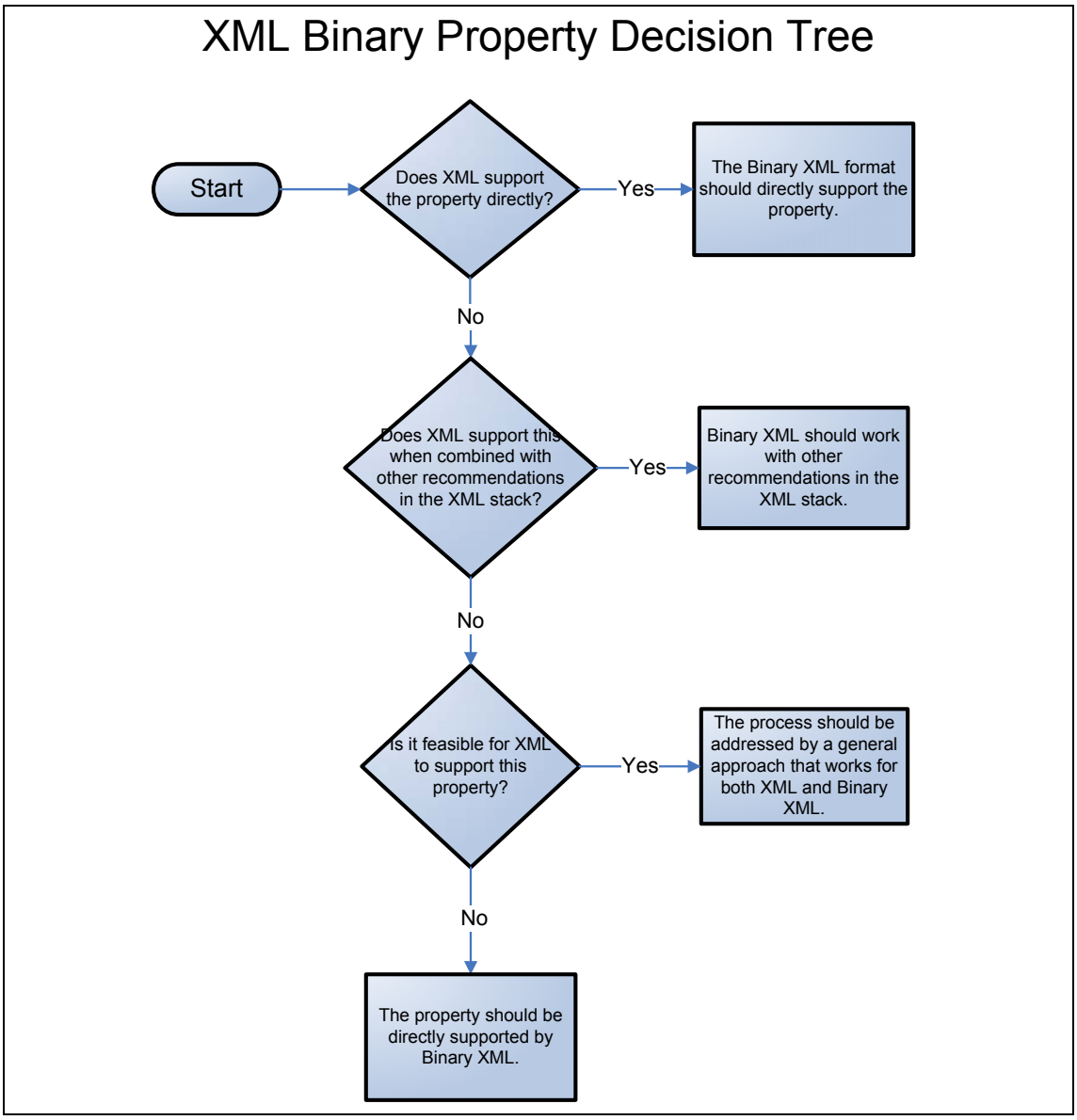

Figure 8. XML Binary property decision tree to determine which properties should be directly supported and which are supported by associate technologies (Goldman & Lenkov, 2005)

However, the working group also recognized that some of the properties listed could be satisfied through other means and did not necessarily need to be an intrinsic property of binary XML. The decision tree, shown in Figure 8, shows the process through which each property was passed. The decision tree is designed to differentiate between properties that should be directly supported by binary XML and those that can be supported by coordination with other technologies in the XML stack

(Goldman & Lenkov, 2005). This ensures that binary XML maintains a close relationship to XML so that future changes to the stack take binary XML into account as well. The output of this process was a list of properties binary XML must support and a list of properties that should be tackled by other technologies working together with XML. The latter of the two lists was then further examined and the properties that binary XML must not prevent from developing were selected for inclusion on the minimum requirements list. These properties, seen in the right-most column in Table 3, do not have to be supported by an implementation but should not be obstructed by binary XML. An implementation should try to contain these properties, but they are not required.

The must support requirements list shown below is then constructed from properties that meet two criteria. They are a "must have" for at least one use case and are needed to be directly supported as indicated by the decision tree. In addition, six properties which did not meet the conditions stated above were kept as they are best practices documented by the W3C and would become requirement for the recommendation regardless (Goldman & Lenkov, 2005). Those are indicated by the W3C column.

MUST Support	W3C	MUST NOT Prevent
Directly Readable and Writable		Processing Efficiency
Transport Independence	W3C	Small Footprint
Compactness		Widespread Adoption
Human Language Neutral	W3C	Space Efficiency
Platform Neutrality	W3C	Implementation Cost
Integratable into XML Stack	W3C	Forward Compatibility
Royalty Free	W3C	
Fragmentable		
Streamable		
Roundtrip Support		
Generality		
Schema Extensions and Deviations		
Format Version Identifier		
Content Type Management	W3C	
Self Contained		

Table 3. Minimum requirements for binary XML Including those that are W3C best practice standards (Goldman & Lenkov, 2005)

c. Conclusions

With these requirements in place, the working group discussed eight current binary XML solutions to see how they matched the minimum requirements as specified above. Each solution had a representative present to answer questions concerning algorithms and functionality. The results were published anonymously so as to prevent an implicit endorsement of one solution. Two of the solutions met all the minimum requirements and two more solutions failed by not supporting two properties. It is important to note that these solutions were developed before the requirements were posted, so any requirements that are met are somewhat incidental.

With this encouraging data the working group made four conclusions about binary XML. They concluded that binary XML is needed, it is feasible, it must be produced by the W3C, and it must integrate with XML(Goldman & Lenkov, 2005).

The need for XML is shown clearly in the varied use cases that were assembled from the technical community. These needs were not being met elsewhere by a single format, but ad hoc solutions. Much care was taken to ensure that the requirements placed on binary XML will meet the needs specified in the use cases.

After examining the eight existing solutions and seeing that two met the minimum requirements with others following close behind, the group concluded that a single solution is feasible—that the twenty-one minimum requirements decided upon are not unreasonable (Goldman & Lenkov, 2005). The group was confident that, with the requirements now published, additional solutions can be designed and developed to meet the specification.

Finally, the working group felt that the W3C must be responsible for standardizing the binary XML solution (Goldman & Lenkov, 2005). To avoid multiple implementations gaining footholds of popularity in their respective areas of use, the W3C must set an international standard. This will protect the interoperability of XML itself as well as its integration into the XML stack. This insures that any future changes made to the stack include binary XML in the consideration as well.

D. EFFICIENT XML INTERCHANGE

Although the normal process for a working group is to start at the beginning with use cases and finish with a new standard, the process for determining binary XML is being altered slightly. Instead of having one group perform all the work, it is being divided into two groups, with the first being the Characterization Working Group. Now that it has fulfilled its chartered purpose and produced its deliverables that group has closed and a new group is being set up to review implementations, conduct testing, and ultimately decide upon the standard. This group is being called the Efficient XML Interchange (EXI) Working Group and the charter is currently being created.

The specifics of the charter for EXI are very important as they will direct the group through the process of selecting a standard. Although there is some contention now about exactly what the charter should specify, it is accepted that the work of the EXI Working Group will be based upon the work performed by the previous working group. The minimum requirements extracted from use cases will be examined and used in conjunction with the measurement methodologies prescribed by the Characterization Working Group. However, these only provide a baseline for examining and testing solutions. Many other issues such as minimum performance criteria need to be agreed upon before the testing begins. Vendors have already notified the W3C of their work on potential candidates, but require the confidentiality of a Working Group to disclose their solutions. As there are many major stakeholders involved with developing a new standard, it is taking some time to complete the charter.

E. ANALYSIS OF XSBC VIA PRESCRIBED CHARACTERIZATION

1. Must Support Properties

The current implementation of XSBC currently fails to meet all of the “must support” properties as outlined by the working group as seen in Table 4, but work is underway to ensure it meets all requirements. Specifically, the fragmentable, schema extensions and deviations, and format identifier properties are not met. The schema extensions and deviation property fails due XSBC’s reliance on an XML schema document. XSBC is inflexible on its stance pertaining to schema use. The specific .xsd file must be co-located with an XML file in order for XSBC to serialize and deserialize

that file. The schema deviation property is included because it provides flexibility by being able to encode files in which the user has no prior knowledge of their structure or type. Future work is planned to free XSBC from such a strict reliance upon a schema. In addition, format version identification is not difficult to implement, but at the time of this writing it had not yet been incorporated. The fragmentable property can also be added to XSBC in the future. Resources need to be available to do this, but the modular design of XSBC allows implementations of additional functionality.

MUST Support	**Fulfills**
Directly Readable and Writable	X
Transport Independence	X
Compactness	X
Human Language Neutral	X
Platform Neutrality	X
Integratable into XML Stack	X
Royalty Free	X
Fragmentable	Planned
Streamable	X
Roundtrip Support	X
Generality	X
Schema Extensions and Deviations	Planned
Format Version Identifier	Planned
Content Type Management	X
Self Contained	X

Table 4. XSBC score table of W3C XBC MUST SUPPORT properties

2. Must not Prevent Properties

XSBC fulfills all the “must not prevent” requirements set forth by the working group. While XSBC does emphasize the small footprint at the cost of processing efficiency, it does not prevent that activity. As it is an open source implementation of binary XML developed in Java, it most certainly does not prevent widespread adoption or implementation cost. The code is free the Internet and Java code is portable, by nature, to multiple operating systems.

MUST NOT PREVENT	Fulfills
Processing Efficiency	X
Small Footprint	X
Widespread Adoption	X
Space Efficiency	X
Implementation Cost	X
Forward Compatibility	X

Table 5. XSBC score table of W3C XBC MUST NOT PREVENT properties

3. Conclusion

XSBC does not completely meet all the requirements set forth by the W3C. However, it still contains capabilities to be explored while the EXI specification is underway. XSBC contains all but four properties, which suggests it has utility in a subset of the use cases. Utilizing a schema is the characteristic that caused XSBC to fail to meet certain properties, but in a scenario where the data is consistently being transmitted in a known format (e.g. tactical data links or X3D) using a schema is not detrimental but an asset. Schemas provide a powerful integrity check for the received documents and in this way provide a first layer of security. In these cases of operating on data with a static format, XSBC provides a realistic solution. XSBC provides an excellent testbed for further experimentation on EXI candidates

F. RECOMMENDED STRATEGIES FOR NATO AND U.S. DOD

Based on the positive outlook from the binary XML working group, the author encourages both NATO and the DoD to actively track and participate in the W3C EXI group. Many advantages for the military lie in the recommendation and successful implementation of a binary XML format as seen from the use cases. While the EXI is in a transitional period it would behoove the DoD to begin its own testing, evaluation, and use of binary XML. XSBC and Fast Infoset are free open source implementations and can be easily obtained. AgileDelta's Efficient XML solution may also be obtained as AgileDelta has performed excellent work for the DoD in this area already.

While engaged in the EXI Working Group, it is recommend that the DoD and NATO ensure forward compatibility for any binary XML solution considered. Practically,

this means that a compression archive be maintained for any binary compression algorithms that could possibly be used in the future. In addition, it is recommended that the DoD and NATO perform further comparison studies to compare effectiveness of legacy customized binary protocols (e.g. LINK16, Distributive Interactive Simulation, etc) with generalized binary compression such as that provided by Fast Infoset, XSBC, or forthcoming EXI format. Although binary protocols are highly tuned and superior compression may not be possible, nevertheless, general compression techniques that provide comparable compression along with XML web services compatibility may well prove to be a better long-term choice.

G. SUMMARY

This chapter covered how the properties for a standard binary XML solution were established. It also inspected XSBC to see what properties are currently being met and where improvements are needed.

THIS PAGE INTENTIONALLY LEFT BLANK

V. NATO REAL-TIME TACTICAL COMMUNICATIONS

A. INTRODUCTION

This chapter provides an overview of selected tactical data links used by NATO as well as an overview of XML technology and how it can be used to further improve the data links.

B. TACTICAL DATA LINKS

1. Introduction

Establishing good communications has always been an essential element of an effective fighting force. The commander overseeing the operation must have the ability to control and coordinate his forces to maximize their effectiveness through a unity of effort. However, good lines of communication not only deliver information down the chain of command but also feed information back up. In order to decide on the best course of action, the commander needs to have excellent situational awareness which includes information regarding both friendly and enemy forces (Howland, 2004). This concept of operations applies not only to strategic level commanders, but also at the tactical level, such as a combat pilot. While only a single individual a pilot's effectiveness is contingent upon his ability to maintain good situational awareness (SA) which can be difficult due the high rate of change of his operating environment.

While voice communications can be used to maintain SA, the potential for miscommunication exists, especially in a fast-paced high-stress combat environment. In addition, voice communications are completely ineffective for transmission of highly precise and dynamic data such as aircraft positions. With serious consequences attached to actions taken in such a scenario, an unambiguous method of communication is vital.

A digital data link provides many advantages over voice communications in regard to this particular type of communication. First, sending digital text is much simpler to implement and requires much less bandwidth than a full-duplex voice channel (Howland, 2004). Second, the information can be continually and automatically updated

which is particularly useful for tracking aerial targets. Finally, digitalization of data also allows for verification of information though an unambiguous format, forward error correction, and encryption. This lays a solid foundation to build upon for reliable and secure communications.

Organizations such as NATO also contain unique challenges as joint forces are not only from varying services but also from different countries (Muller, 2003). Not only are there differences in communication equipment, protocols, and standards, but the end users may not even speak the same language. This only underscores a need for a standardized unambiguous format and method for communication. The current family of tactical data links was meant to be a solution for all of the afore-mentioned issues.

	United States	NATO
LINK 1	N/A	STANAG 5501
LINK 4 / TADIL-C	MIL-STD 6004, MIL-STD 188-203-3	STANAG 5504
LINK 11 / TADIL-A	MIL-STD 6011, MIL-STD 188-203-1A	STANAG 5511
LINK 11B / TADIL-B	MIL-STD 6011, MIL-STD 188-212	STANAG 5512
LINK 14	N/A	STANAG 5514
LINK 16 / TADIL - J	MIL-STD 6016	STANAG 5516

Table 6. United States military and NATO standards for tactical data links (Air Land Sea Application Center, 2000)

NATO currently uses LINK 1, 4A, 4B, 11A, 11B, 14, and 16 for its tactical data link needs with LINK 11B and LINK 16 are the most prevalent and bear the bulk of the workload (Downs, 2005).

2. LINK 11

LINK 11 is one of the oldest yet employed tactical data links. It is still used by several countries within NATO including the United States where it is known as TADIL-A/B (Howland, 2004). TADIL-A, the older of the two versions, is based on 1960's technology and is used for airborne communications. It employs digital radio signals in both the HF and UHF ranges. HF propagates a ground wave which allows for communication out to ranges of 300 nautical miles while UHF limits users to Line of

Sight (LOS). This establishes a maximum UHF range of 150 nautical miles for communications between a surface vessel and an aircraft (Downs, 2005).

a. TADIL-A

TADIL-A operates as a netted communication architecture by establishing a simple polling system. Every unit can be classified as a Net Control Station (NCS) or a Participating Unit (PU), however only one Net Control Station exists on a particular net (Downs, 2005). The NCS is responsible for sequentially polling each PU within the net and for managing the resources of that particular net. During a roll call procedure, the NCS polls the PU's in a particular order and only then can a PU transmit data to the NCS. The NCS, with the updated data from the entire net, serves as the hub for the network and is able to broadcast the data to all PU's on the net. LINK 11 operates generally at two data rates: 1364 bits per second (bps) or 2250 bps (Downs, 2005). These data rates include error detection and correction bits, but its network capacity is very small by current standards; the system is now almost thirty years old.

Perhaps the most vital function LINK 11 makes possible is the creation of a Common Operational Picture (COP) (Downs, 2005). The NCS receives track data from the sensors for each PU, compiles that data, and correlates the tracks. This ensures that two tracks of a single contact are fused and prevents the creation of a contact that does not actually exist. LINK 11 is also used for various command and control functions to include transferring control of aircraft, maintaining aircraft status, and issuing commands.

b. TADIL-B

TADIL-B is an upgraded version that of LINK 11 with basic changes in architecture and increased data rates. TADIL-B is used primarily by ground units for tracking and controlling air contacts. With the upgraded link, the designers have broken away from the aging polling architecture and TADIL-B boasts a dedicated point-to-point architecture. This also allows for full duplex sending and receiving. Along with these

improvements comes increased bandwidth. LINK 11B nominally operates at 1200 bps, but has the ability to transmit at multiples of 1200 even past 4800 bps.

Both LINK11 and LINK11B share the same message format. All the information sent over the data links must be one of the fifty-three standard messages defined in STANAG 5511 as the M series (Downs, 2005). This eliminates any ambiguity in the message content as well as allows for automatic processing of the messages. However, the actual binary readers and writers that convert the streaming radio data into the messages are proprietary. This isolates the system as the data is not in a common format which prevents it from being readily available to other systems.

3. LINK 16

LINK 16 does not stray far logically from the traditional idea of a tactical data link. It exists as a communication system that provides real time command and control capability. However, it offers a myriad of improvements over its predecessors as the newest of the TADIL family and incorporates newer technology. One of the most important improvements is the reduction in physical size of the module. This allows for tactical users such as fighters to be connected via LINK16 and not just larger C^2 aircraft such as the E-8C Joint STARS (Pike, 2000).

The most significant architectural change is the removal of the NCS that was present with LINK 11. A decentralized architecture is used by LINK 16 with multiple users implementing time division multiple access (TDMA) schemes; meaning information is broadcast to units where that information is relevant. All participating units in a network are categorized into a Network Participation Group (NPG). Each NPG is identified by its information needs or role within the battle space. The unit coordinating real time surveillance does not need to receive the air traffic control vector information as it is a waste of resources to do so. As a result, units only transmit data to others within their NPG. This eliminates unnecessary network traffic as well as the need for a central control node which is a considerable vulnerability that exists with

LINK11. Other improvements upon the existing TDL architecture include greater overall throughput, electronic countermeasures resistance, an automatic relay system, and the addition of secure voice channels.

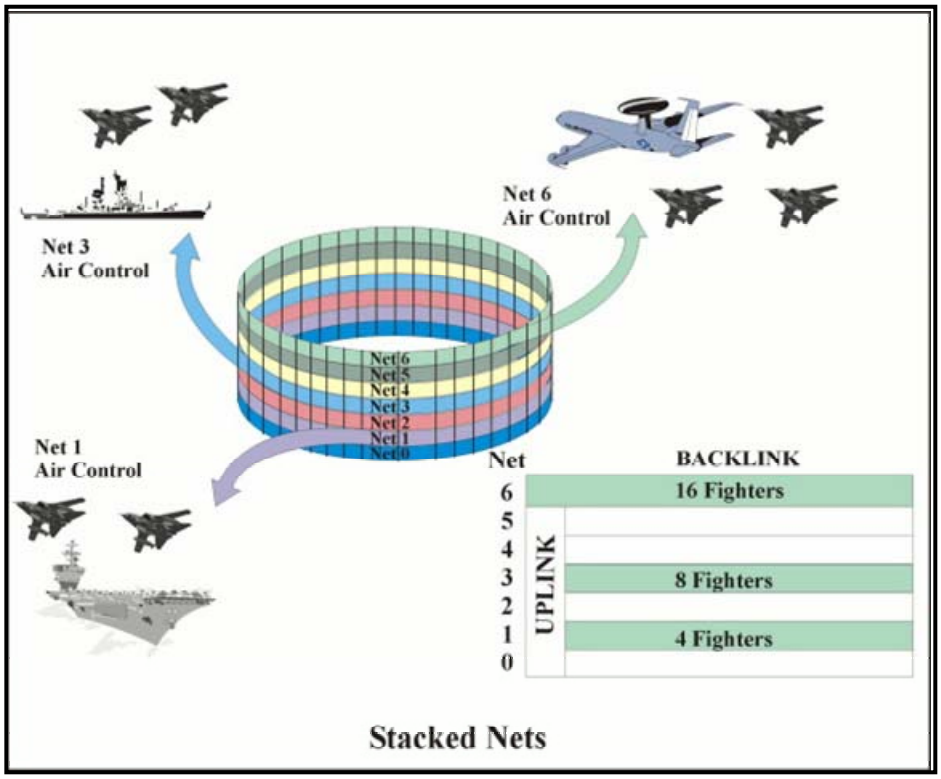

Figure 9. LINK16 stacked net architecture showing the separation of nets into logical communication groups (Downs, 2005)

LINK 16 operates in the UHF spectrum which provides more frequency bandwidth than LINK 11. This provides additional raw spectrum for higher throughput, but reduces the range of effective communications. To combat this limitation of UHF frequencies, LINK 16 features an auto relay system for units that are beyond the line of sight. Units that are within range receive the message and then forward it to those unable to receive the original transmission (Pike, 2000). This decentralized architecture is similar to having each unit act as an NCS.

LINK16 also further maximizes the frequencies available by implementing a frequency hopping scheme. This has multiple benefits including an increase in

throughput, anti-jamming capability, and the ability to stack nets and create entirely separate networks. A net is created by designating a specific frequency-hopping pattern. Then the nets are assigned to various platform groups that require inter-communication ability as displayed in Figure 9. This allows communication within the net without interfering with other platforms assigned to other nets. As a result, platforms on different nets can transmit during the same timeslot, eliminating interference, as they are operating on different frequencies (Downs, 2005).

These multiple nets contained in a stack constitute a logical network. One can easily switch between nets within the network, but to switch networks requires a re-initialization of the physical LINK16 module which is much more involved than switching nets. Switching nets in necessary as one can only belong to a single net at a time. This provides a framework for organizing communications among users with a common mission and providing security by separating networks (Downs, 2005).

The message format for LINK 16 is similar to that of fixed-length format found in LINK11. NATO's STANAG 5516 and the US MIL-STD-6016 define the messages as the J series (Downs, 2005). Every J series message consists one or more 75 bit "word". There are three types of words: the initial word, extension word, and continuation word (Downs, 2005). If a message with multiple words is sent, the words must arrive in the aforementioned order. The number of words required for the message is dependant upon content of the message. The most common combination is an initial word followed by an extension word. The initial word is always required as it give the basic information of the message as well as the number of extension and continuation words that are to follow it. The extension word contains additional detailed information that could not be contained in the initial and the continuation word contains amplifying information when available.

C. NATO XML PROGRAM

NATO Consultation, Command and Control Agency (NC3A) is a technical institution that seeks to provide NATO with the latest command and control technology. NC3A, along with most major software developers, took notice as XML emerged onto

the market and rapidly gained popularity as it proved to be an excellent solution to many data management problems. With the knowledge that NATO is a very complex and data rich organization, NC3A recognized the value and advantages XML had to offer especially in military message formats, semi-structured documents, and merging heterogeneous databases (Muller, 2003).

Early on NATO realized the need for a common message format for communication among the 19 member nations. Subsequently, the Allied Data Publication No. 3 (AdatP-3) and the United States Message Text Format (USMTF) were created (Muller, 2003). These messages are structured and are used for not only tactical messages but every type of communication within a military structure. Needless to say, this format, originally designed for teletypewriters, can be mapped to an XML structure with relative ease.

NC3A also is considering the feasibility of using XML to markup the vast amounts of information that flow in and out of NATO on a day to day basis (Muller, 2003). With metadata embedded in every electronic document, the efficiency involved in searching, sorting, and correlating data could greatly be improved. Similarly, NATO has access to multiple databases from many different sources. Each database contains unique data models and structure and interoperability among them is currently cumbersome. However, a conceptually consistent XML schema can provide a common framework to aid in this data exchange.

1. NC3A Workshop

As a result of the aforementioned potential applications of XML, NC3A held an XML Workshop in November of 1999 in order to formally investigate the direction NATO should take concerning XML (Muller, 2003). The workshop included military and civilian experts. The workshop sought to discover potential uses for XML within NATO and their associated organizational and financial benefits. In addition, the workshop focused on the current state of military development of XML-enabled capabilities among NATO nations in addition to the status of development in the civilian industry.

Those involved with the workshop concluded that XML is indeed an important technology for NATO and its members and that early action is required in order to prevent the benefits offered by XML to be muddied by independent and uncoordinated development efforts. Specifically, recommendations were made to begin educating decision makers within NATO of the potential XML offers while monitoring the affects XML has on data management as it is implemented among the Nations. In addition, the workshop proposed the founding of a NATO XML Registry and Repository Service to act as a coordinating body for development (Muller, 2003). Finally, it was recommended that NATO should provide a representative to international standards organizations such as W3C and OASIS in an effort to help shape XML standards so that the many military use cases will not be overlooked. In this way, the workshop hopes to provide input into the standardization process and so shape the future of web service technologies.

2. NIRIS

NIRIS first stood for the NATO Interoperable RASP Information System with RASP meaning Recognized Air and Surface Picture (Howland, 2004). It has been renamed to reflect its newest upgrade capabilities; Networked Interoperable Real-time Information Services. This program is currently an example of the recommendations made by the NC3A Workshop. What began as an ad hoc solution is now becoming potentially a powerful C^2 tool for NATO. This section details the creation and functionality of NIRIS with a focus on XML integration.

NIRIS began as an ad hoc solution to a request made by Supreme Headquarters Allied Powers Europe (SHAPE) during NATO's involvement in Bosnia in 1994-1995(Howland, 2004). The NATO strategic level commanders in Brussels wanted to be able to view the LINK11 air track data occurring within theater. More specifically, SHAPE wanted the tracks from each individual combined air operations centers (CAOC) to be combined to create a common operating picture of the air operations. NC3A fulfilled the designated requirements, but the solution required specialized equipment and was far from extensible.

NIRIS continued to be a useful tool and as a result a second version of NIRIS was developed in 1999. However, the entire program was re-coded from the ground up in Java retaining only a few segments of the original C code. The retooling of NIRIS provided improvements to include the ability to record and replay data as well as an API to allow other programs to access and use the data stored by NIRIS.

It was not until 2004, though, that it was decided to introduce XML support and capability into NIRIS (Howland, 2004). This third version was fueled by the desire to shift NIRIS from its current stovepipe paradigm into a web service architecture. Currently, version 3 contains three important modules that provide the principal functionality of NIRIS. These modules are named TITO, TIXO, and XITO.

a. TITO

TITO serves multiple functions and as a result stands for TDL-in, tracks out; or tracks-in; TDL-out; or TDL-in, TDL-out. This module decodes any LINK1, LINK11, or LINK16 message and provides an API for other programs to be able to access the data contained in the messages as well as generate their own. TITO can also translate among the different LINK formats according to the NATO Standardization Agreements (STANAG) and then forward the message. The process of coding TITO to appropriately parse the 3000 different TDL messages per the STANAG's was accomplished by a new state-of-the-art code generator (Howland, 2004). This process was so precise and thorough that it discovered ambiguities among the STANAG's which are currently being resolved.

b. TIXO

TIXO which stands for tracks-in, XML-out is a module that adds considerable flexibility and power into NIRIS by providing a core requirement for web services: the data in an XML format. Currently, TIXO takes the track data from TITO and then placing data into an XML format. Once this is accomplished the data is available for the use by web applications. Currently, TIXO supports three data formats in XML: NVG, XLTF, and TDL-XML (Howland, 2004). The NATO Vector Graphic (NVG)

format is a specialized standard used in creating military map overlays. In generating an NVG message, TIXO extracts the position, velocity, and track type from a TDL message via TITO and ads information so that the correct MIL-STD 2525B symbol is associated with the track(Howland, 2004). The Extensible Light Track Format (XLTF) is similar to NVG in that it extracts information deemed important from a TDL message, but differs in that it is extensible and uses can add optional information about the track for specialized purposes. The TDL-XML format is direct encoding of the original TDL message. The advantage of this format is that it provides the same data fields as the original track information.

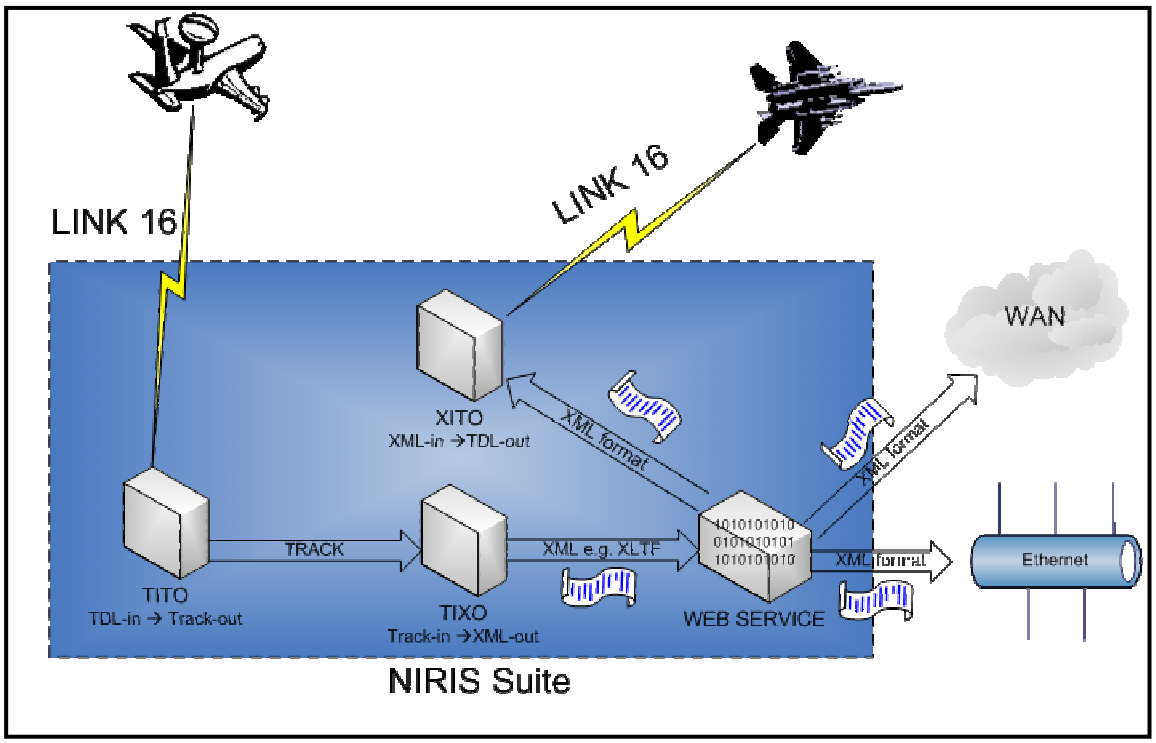

Figure 10. LINK16 round trip demonstration via the NIRIS software suite along with providing web services of LINK16 data

c. XITO

XITO is the newest module in NIRIS and provides a round trip capability into the NIRIS bundle as shown in Figure 10. XITO takes an XML encoded message and translates it into the original TDL encoding to be transmitted back out to tactical

users. In this way NIRIS will allow legacy equipment to interact and take advantage of emerging web services. The process of NIRIS subscribing to a service, performing the conversion into a traditional TDL format, and forwarding the message, will be completely transparent to the legacy link terminal. Ideally, all the TDL's in the future will operate using come variation of XML encoding, but until then, XITO will serve as the gateway between legacy TDL's and a service oriented architecture. Figure 11 shows the roles NIRIS fills in a real-time data web service.

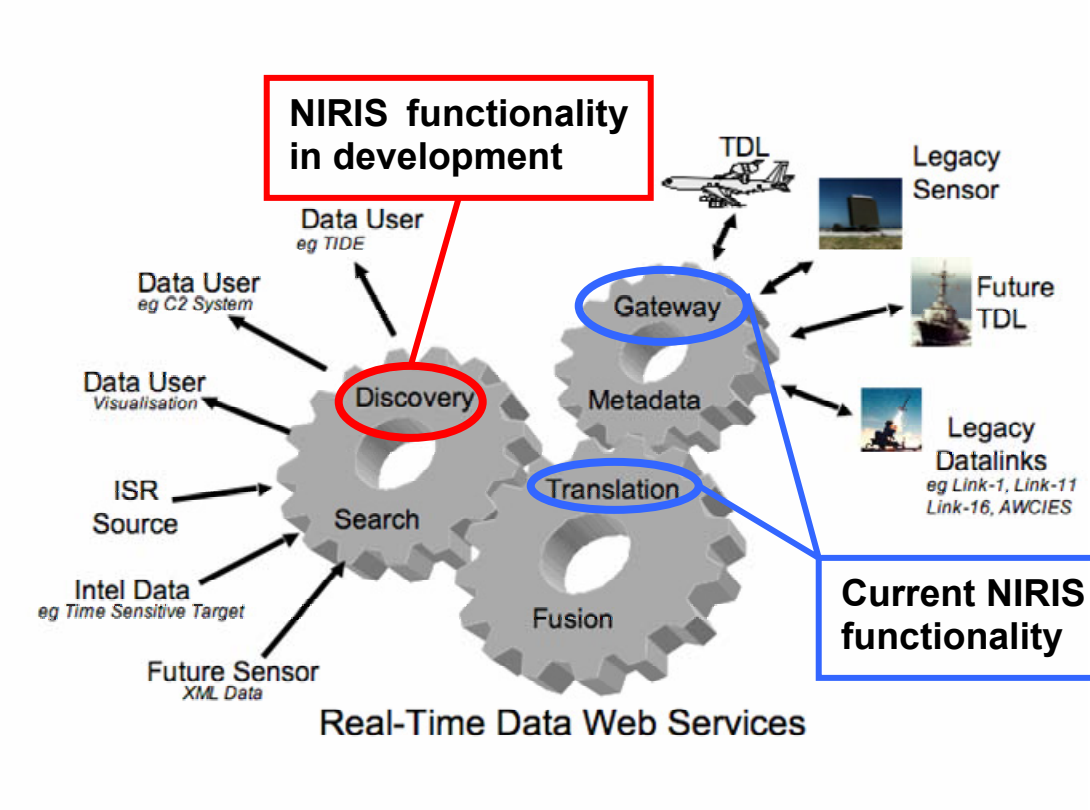

Figure 11. Web service architecture needs currently met by NIRIS and the future needs that NIRIS will meet (Howland, 2004)

d. Experimentation

With all the internal modules built into NIRIS, experiments have been conducted to test module functionality within the context of running an actual web service. The most recent experiments have focused on providing web services coupled with the TIXO module. Two experiments were conducted which required the creation of two new web services that run on the NIRIS framework: TIXO-TIDE and TIXO-STGP

(Howland, 2004). The TIXO-TIDE web service functionality is included in the latter experiment which also contains improvements, but is included to show the evolution of products.

(1) TIXO-TIDE. The first of two experiments was to provide a web interface for an existing tool found in the TIDE WISE framework that queried LINK16 data and displayed it on a geographic overlay. The resulting web service organic to NIRIS was named TIXO-TIDE and provided LINK16 data to TIDE WISE in the NVG format (Howland, 2004). . While NIRIS is able to connect to active LINK16 nets and provide data in real time, for this experiment two hours worth of recorded tracks were played back to simulate live interaction. The goal of the experiment was to create a web service client that received NVG messages unaware of the underlying LINK16 message specification. Figure 12 shows the TIDE WISE client displaying the data.

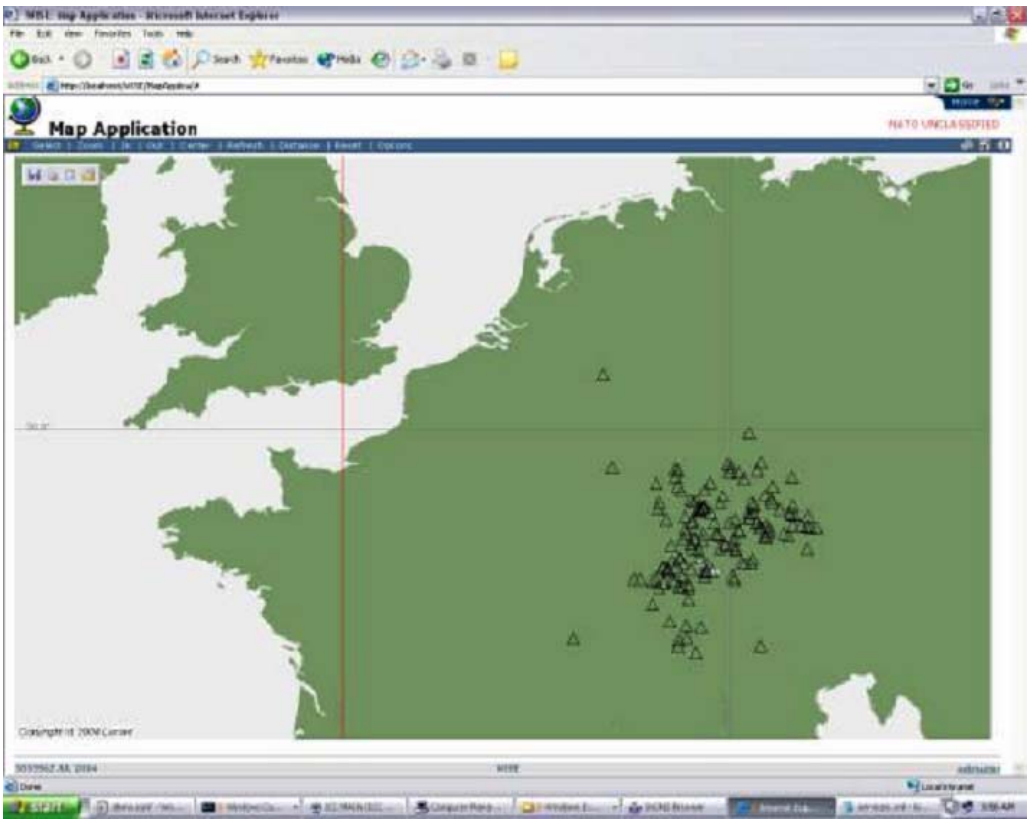

Figure 12. A TIDE WISE client running in a browser displaying NVG data from a recorded LINK16 feed (Howland, 2004)

While lacking in the detail that will eventually be required by tactical users, the TIDE WISE client successfully demonstrated the feasibility of creating a thin client that relies on a web service to act as a gateway for a TDL as well as performing any necessary conversions.

(2) TIXO-STGP. The second experiment to be discussed is the implementation of the existing STGP system (Shared Tactical Ground Picture) into a web service. STGP is a current program created to enable NATO nations to create a common operating picture from multinational data sources. This experiment involved observing the difficulty in adding an STGP interface into NIRIS to provide STGP client access to the track data store (Howland, 2004). The web service was designed to serve data in the XLTF format to clients on either a query based or subscriber based system. The query based client receives the full TIXO data output per each request while the subscriber initially receives all requested data and then is supplied with updates. The construction of the TIXO-STGP web service was accomplished relatively easily in a manner of weeks and a demonstration of the capability was subsequently executed in January of 2005 under the title Shared Tactical Group Picture Demo 6 (Howland, 2004). STGP's core services (security, discover, etc) were running on servers located in the United States, while other services were located physically in United Kingdom, Norway, and the Netherlands. NC3A, located in The Hague, ran client applications that utilized each of these services via a secret VPN through the Internet. This is shown below in Figure 13.

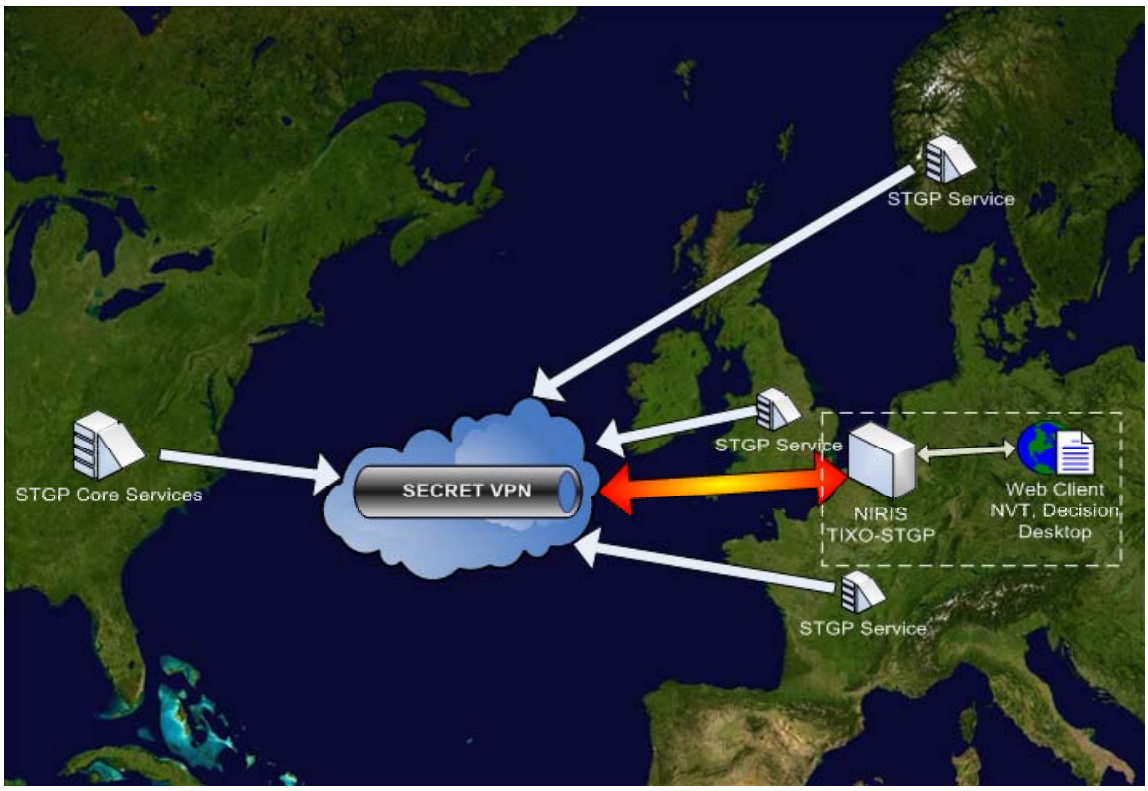

Figure 13. TIXO-STGP experiment set-up showing connectivity of global web services through a SECRET VPN

Not only was this an excellent display of world wide networking, but it was also the first time a secret VPN had been used to securely tunnel through the Internet (Howland, 2004).

Four STGP clients developed in the UK, US, Norway, and by NC3A were tested. The LINK16 air track data gathered from NIRIS along with data from other sources can be see as dots located on British Decision Desktop tool displayed in Figure 14 (Howland, 2004). The other dots include data garnered from the other web services. This exercise showed the flexibility in NIRIS to accommodate other web services as well as include additional services of its own. It also served to demonstrate the flexibility and robust capabilities offered by a service-oriented architecture. The process of coordinating the exact addressing and routing issues normally required for establishing a such a network were handled automatically through search and discovery services (Howland, 2004).

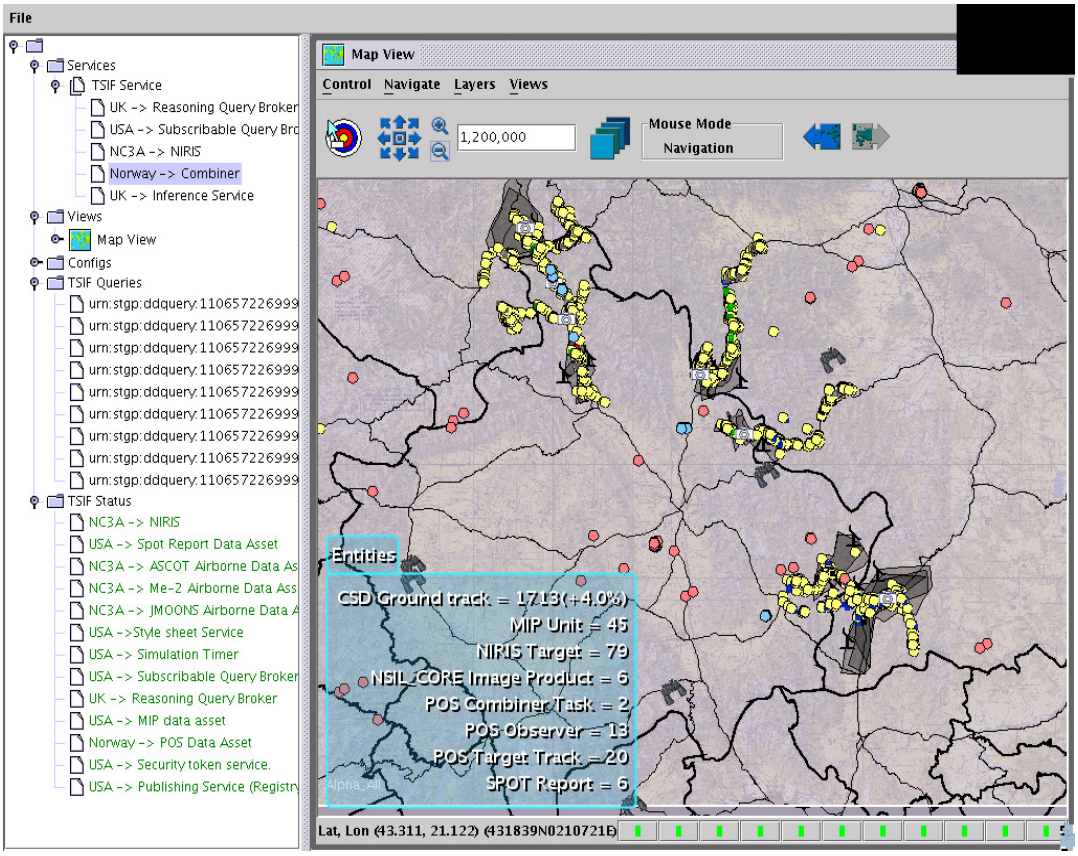

Figure 14. Decision desktop display displaying data pulled from international web services in the UK and Norway (Howland, 2004)

D. XSBC ADAPTATION TO NIRIS

1. Motivations

NC3A has been quite successful in the development of the NIRIS suite, but has sought to improve the performance even more. It was recognized XML's major drawback in this context is its verbose nature as the memory needed to represent text is most often larger than its binary equivalent. This is particularly an issue when transmitting the data over IP networks. Future advancement of TDL might even involve replacing current TDL messages with XML equivalents to be transmitted to participating platforms. Sending smaller messages allows for reduced latency across the network as well as maximizing limited network capacity. Due to design of many TDL's to be ECM-resistant, the overall usage of available bandwidth suffers tremendously as compared to the increased data transfer rates which cable internet users are accustomed. In response to this, NC3A began seeking binary XML solutions. A possible solution was considered in using a dynamic schema coupled with the Abstract Syntax Notation One

(ASN.1) encoding algorithm, but this proved to be far more complex than anticipated. Representatives from NC3A first learned of the binary XML work performed by the MOVES Institute of NPS during an initial meeting at the Interservice/Industry Training, Simulation & Education Conference of 2004. NC3A then contacted NPS MOVES the following year with a request to collaborate on XSBC's implementation into NIRIS.

2. Implementation

Since the TIXO module inside NIRIS outputs an XML document this was deemed the logical place to add XSBC's capability. The TDL messages wrapped in XML, which are defined by a specialized schema, are received by XSBC from the TIXO module and are subsequently encoded and serialized for transmission across the network. Upon delivery, the NIRIS architecture deserializes the stream via the XSBC deserializer. At this point the data is completely abstracted from XSBC and NIRIS operates without any knowledge of the binary encoding. Hence, NIRIS can forward the data to XML enabled web services or to the XITO module for output to legacy displays requiring TDL data. In this way XSBC integrates easily into NIRIS with minimal intrusiveness. This is illustrated in Figure 15.

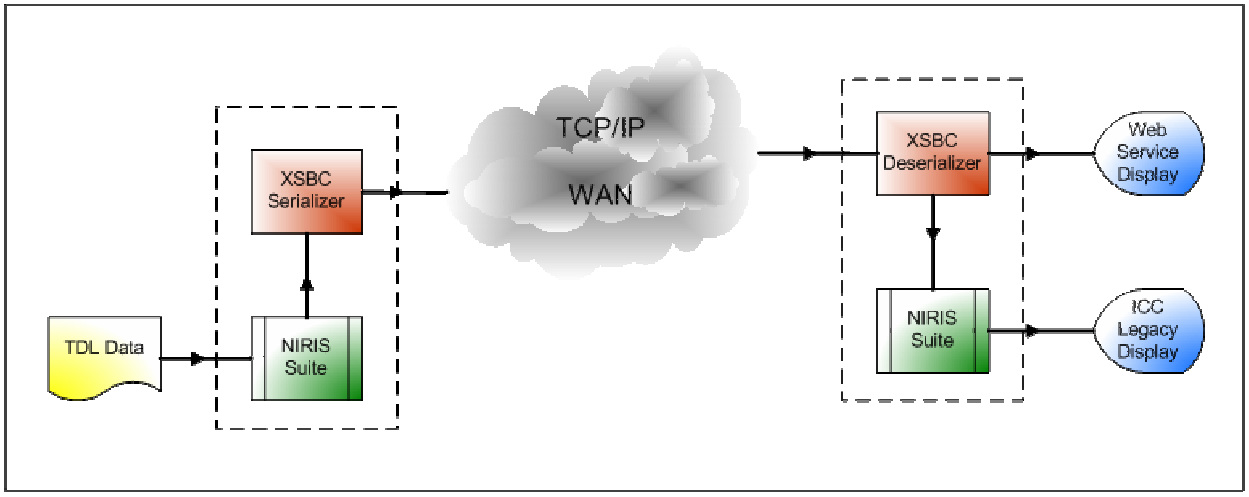

Figure 15. XSBC integration into NIRIS showing a mandatory TIXO interface and optional XITO Interface. Also shown is the binary format's transparency to the user

a. Issues Discovered

The integration of XSBC into NIRIS was performed fairly easily as both programs are written completely in Java. However, a few issues arose exposing current limitations in XSBC. First, it was discovered that XSBC is currently designed to operate with a single namespace referencing a single schema. This became readily apparent during testing as the as Link 16 J-series XML messages containing references multiple namespaces. This characteristic was duly noted and then, as a work-around, an XLST document was created and utilized to strip away the multiple namespaces references from the test messages. It was also discovered that the degree symbol "º" used in latitude and longitude position data resolved to "?" in the deserialized XML file. A similar issue involved the double quotation mark resolving to "&quot". It was discovered that the "&quot" string actually follows an obscure XML convention due to lack of standardization in recognizing UTF-8 special characters. Both of these issues can be corrected by coding the exact UTF-8 hexadecimal values representing these special characters in place of the actual special character in the original or generated XML document, or using the simpler text-based character-entity substitutions such as "&#176" for the "º" symbol (Raggett, Le Hors, & Jacobs, 1999).

Despite the issues mentioned above, XSBC was integrated successfully and tested using recorded LINK16 test data. However, a few items are discussed as future work. The most obvious need is a retooling of XSBC to accommodate multiple namespaces. Until then, XSBC cannot effectively be incorporated into NIRIS as the bulk of the message traffic contains references to multiple namespaces. The other work item of special character resolution is of lesser concern but provides improved efficiency of the TIXO-XSBC interface. Currently, TIXO's output product is XML in the form of a DOM tree. XSBC is not currently able to serialize a DOM tree and as a result, code had to be written to make JDOM call to translate TIXO's output DOM into an actual XML file. Implementing a StAX interface into TIXO would increase efficiency as it is a streaming interface and requires considerably less memory than DOM. Future work in measuring performance needs to examine data binding as well as serialization/deserialization to network streams and files.

E. SUMMARY

This chapter describes the steps NC3A has taken towards producing a web service oriented architecture through the conversion of tactical data link into an XML format. It also examines some of the experiments conducted by NC3A as well as documenting initial NPS integration of XSBC into the architecture.

VI. EXPERIMENTS, DATA COLLECTION AND ANALYSIS

A. INTRODUCTION

This chapter gives specific data on the compression and parsing performance of XSBC and Fast Infoset. GZIP is also used as a follow-on for added compression and those results are shown as well. An analysis of the performance characteristics is also given.

B. BACKGROUND

The original algorithm for XSBC was primarily developed by Don McGregor and Don Brutzman and documented in 2003 by Ekrim Serin under the name of Cross Schema Format Protocol (XSFP). Serin's work contained useful data showing the time to generate and serialize protocol datagram units for the purpose of supporting simulations in networked virtual environments. The thesis also showed the utility of using XSBC in compressing X3D documents. However, due to ongoing development, a systematic approach to measuring the exact performance characteristics of XSBC had not been conducted. As XSBC is a candidate not only for integration into NATO but also as a W3C binary XML solution, its performance profile needs to be known.

C. OVERVIEW

Since binary XML is designed to alleviate network capacity limitations, one of the most important aspects of XSBC to examine is its compression performance. Sending smaller sized files over the network mean the file as a whole is received more quickly which frees up network resources so that more packets can be sent to maximize efficiency. Related to bandwidth is network latency, or the time it takes for a packet to traverse the network.

Another factor to consider is the amount of time it takes for XSBC to parse and serialize an XML file before passing it off to be streamed across the network. This is important for cases that involve real time or near real time data transfer.

D. METHODOLOGY

Serin's original functioning java code was given to Yumetech, an application development company, for modularization and to bringing the code up to industry standards for design. This code is available online via CVS (see appendix C for details). Packaged along with the source are simple example applications to demonstrate some example uses of XSBC as well to demonstrate how to implement it. One of these applications is the comparison panel covered in the previous chapter. This tool was used to gather parsing and compression data on eighteen different XML files of steadily increasing size.

This was done to observe the performance on varying file sizes. The same data was also gathered on the same files, but Fast Infoset was used in lieu of XSBC. For both encodings, non-lossy compression was used.

Latency, another important metric, is related to bandwidth, is. It can be calculated with a known file size and bandwidth as seen in Figure 16.

$$Latency\ (\text{sec}) = \frac{File\ Size\ (kb)}{Bandwidth\ (^{bits}/_{\text{sec}})}$$

Figure 16. Latency equation with known file size and bandwidth

The latency of the entire system can be approximated by adding the network latency to the parsing and deserialization times. However, the best way to measure system latency is to actually serialize an XML file, send it across a network, deserialize it, and measure the overall change in time for the process. Using the sample applications packaged with XSBC as a reference, simple client and server applications were constructed to take such measurements. The code is given in Appendix C. While the theoretical values can be calculated, this application is a useful tool for those trying to meet certain performance goals. One can load an arbitrary number of test files into the appropriate directory, and the server will serialize and send the files over the

network to the client where latency, parsing, and deserialization times are gathered placed into three separate files. One can then examine the data to see if the goals set for the network or test files are met.

E. EXPERIMENT

The Comparison Panel code was downloaded, compiled, and tested on a laptop running the Windows XP Professional operating system, with a Pentium4 2.0 GHz processor, and 512 MB of RAM. This was the machine used for all the tests.

Kilobytes	File Name
1	simple-inherit.xml
2	espdu.xml
3	auvmissionOutput00.xml
4	auvmissionOutput01.xml
5	auvmissionOutput02.xml
6	auvmissionOutput03.xml
7	auvmissionOutput04.xml
8	auvmissionOutput05.xml
9	auvmissionOutput06.xml
10	auvmissionOutput07.xml
20	uuvmissionOutput17.xml
30	uuvmissionOutput16.xml
40	uuvmissionOutput15.xml
50	uuvmissionOutput14.xml
100	uuvmissionOutput13.xml
200	uuvmissionOutput12.xml
300	uuvmissionOutput11.xml
400	uuvmissionOutput10.xml
500	uuvmissionOutput9.xml
550	uuvmissionOutput8.xml

Megabytes	File Name
1.0	uuvmissionOutput7.xml
1.5	uuvmissionOutput6.xml
2.0	uuvmissionOutput4.xml
2.5	uuvmissionOutput3.xml
3.0	uuvmissionOutput2.xml
3.5	uuvmissionOutput1.xml

Table 7. Approximate sizes of files used in XSBC and Fast Infoset compression tests to show performance over a range of file sizes

Table 7 shows the approximate sizes of the eighteen test files. For each file the parse time was recorded as well as the file size with only GZIP used and then the file size of the binary encoding plus GZIP.

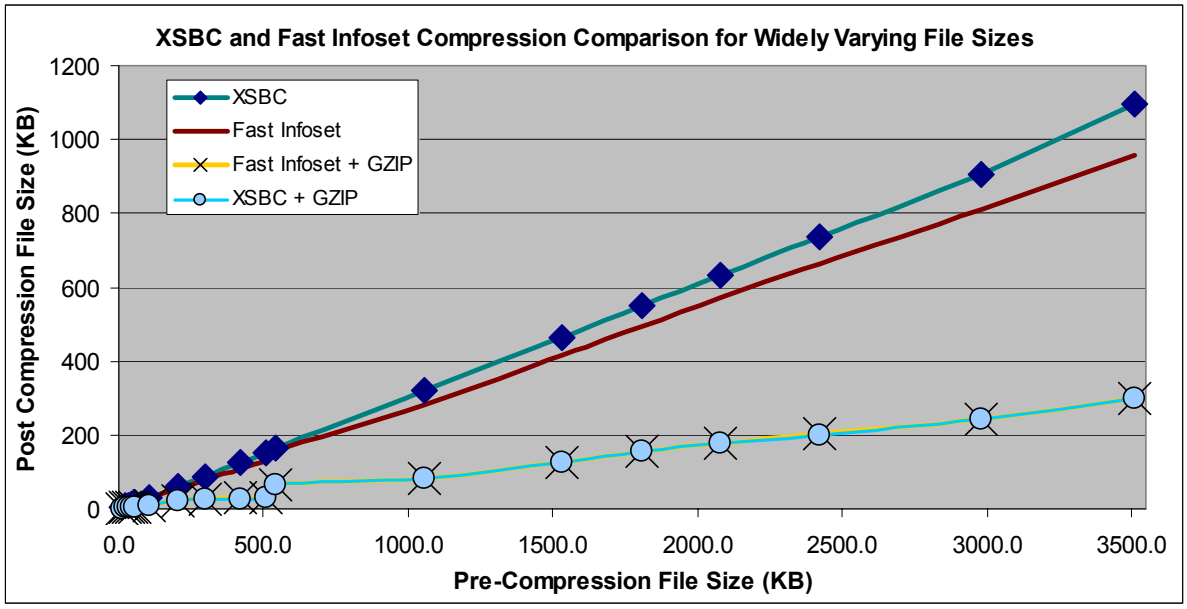

Figure 17. XSBC and Fast Infoset compression comparison

Figures 17 and 18 show the performance of all four variations with 17 giving a close-up on files smaller than 10 KB. As one might expect, as the size of the file increases so does the size of the compressed version. The slope of the line indicates the effectiveness of the compression. A lower slope signifies a better compression algorithm. Also, note that the slope of the each line is nearly constant. This shows that the compression ratios are relatively consistent among varying file sizes.

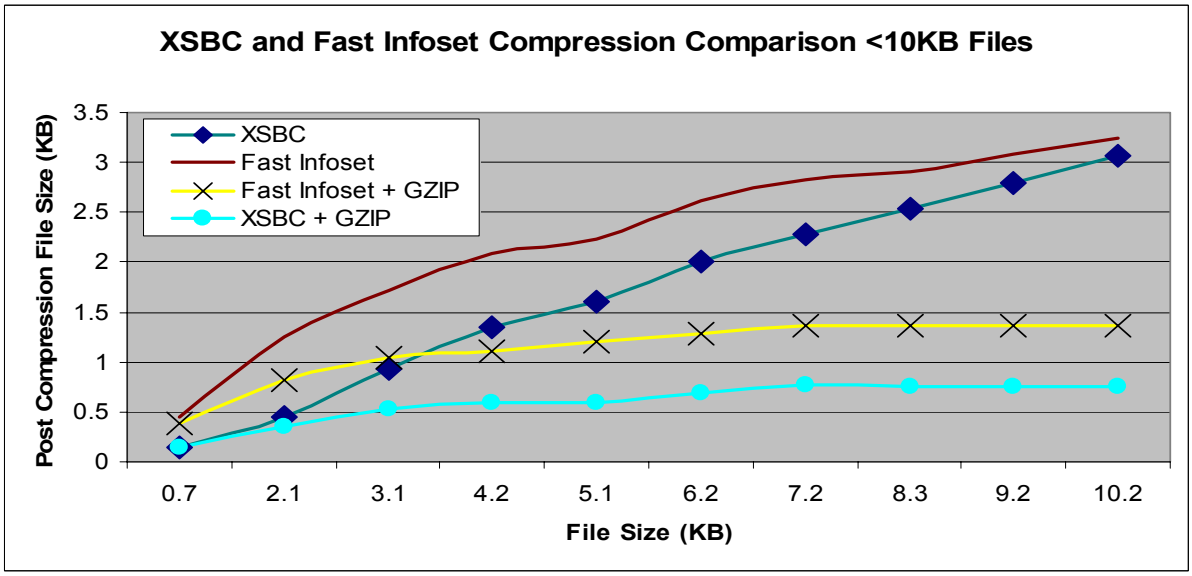

Figure 18. XSBC and Fast Infoset compression comparison for files less than 10 KB

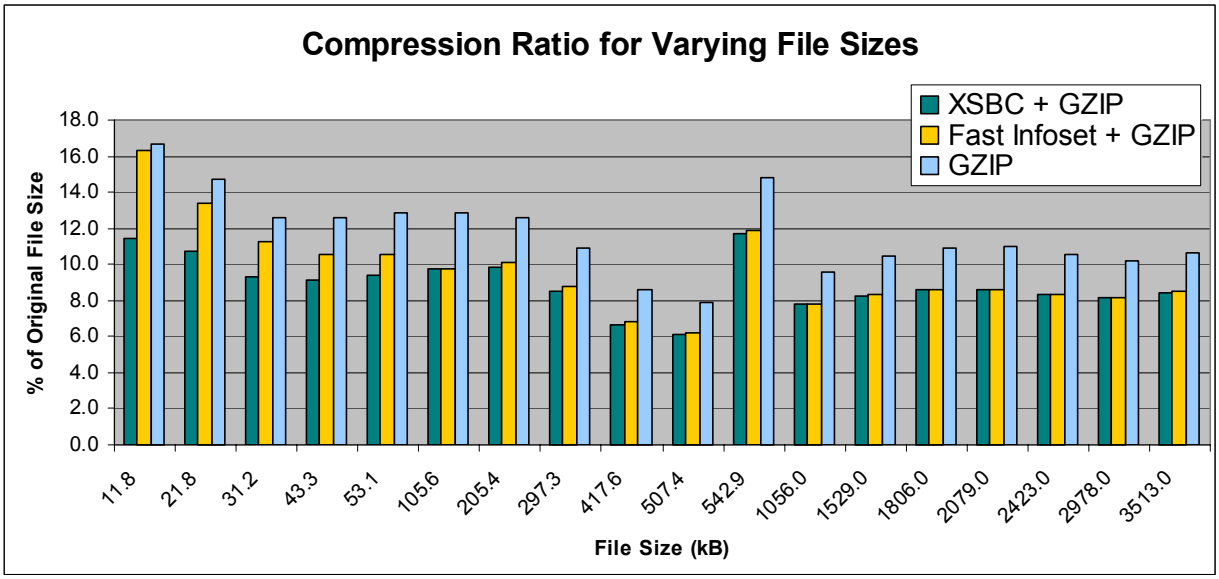

Figure 19. Compression ratio comparison for encodings with GZIP

Figure 19 shows the compression ratio for each of the file sizes, but only for GZIP and the encodings with GZIP added. Figure 20 shows the compression for files less than 10 KB. This allows one to determine which file sizes are capable of maximum compression. XSBC with GZIP is typically able to produce the smallest file size.

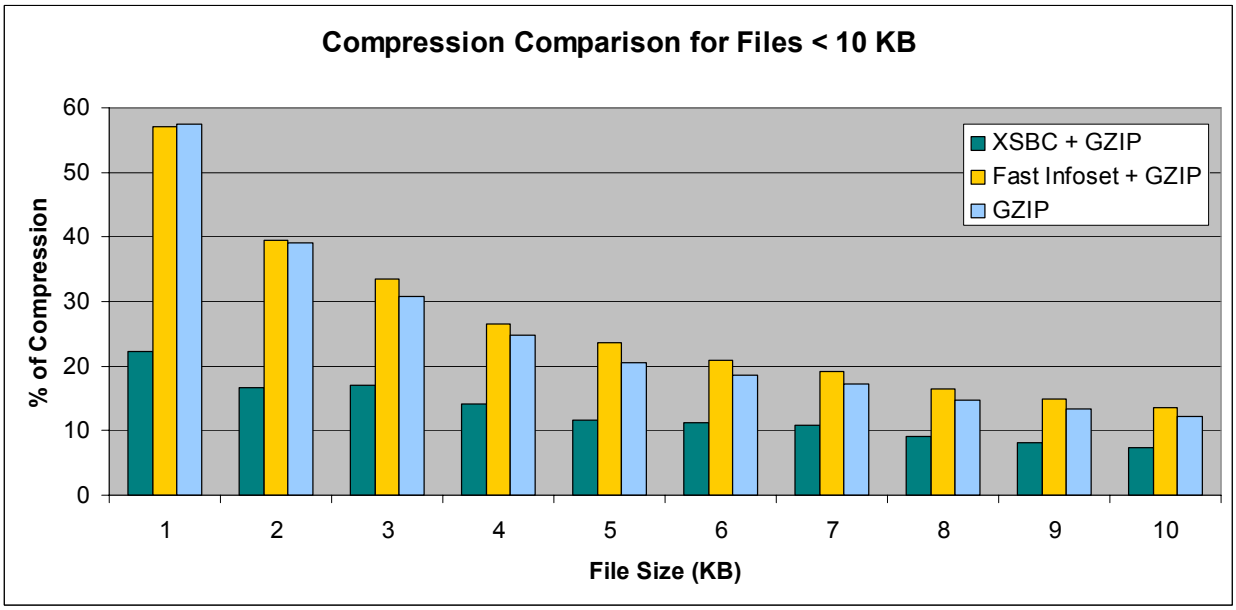

Figure 20. Compression ratio comparison for encodings with GZIP for files less than 10KB

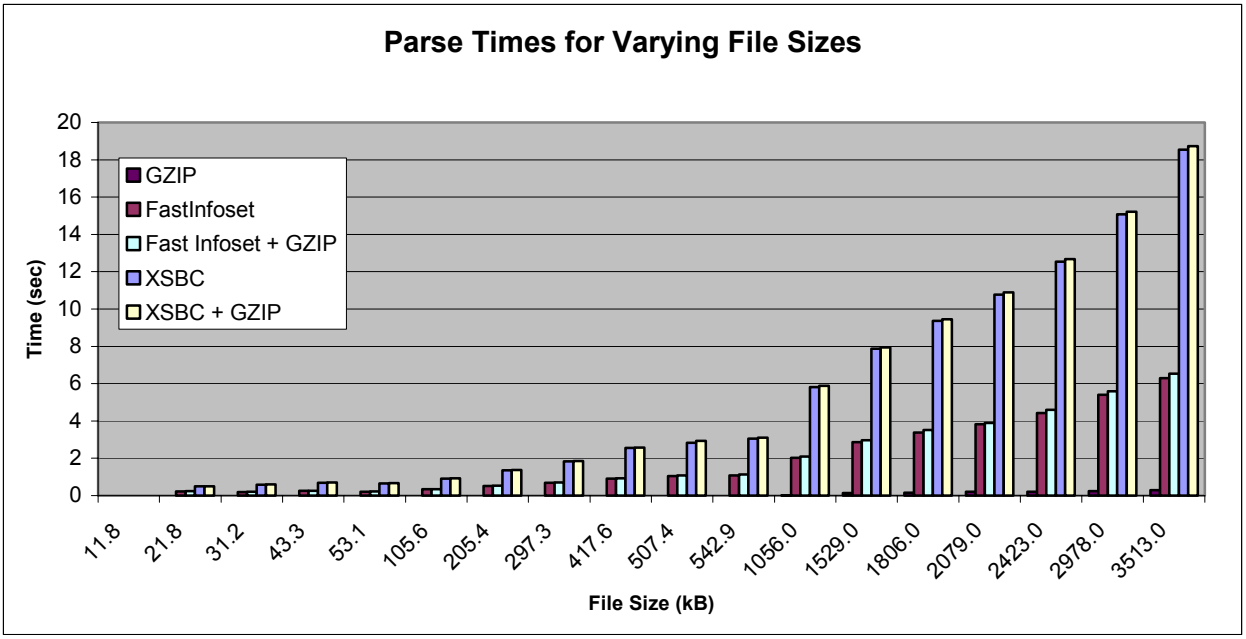

Figure 21. Parse Times for XSBC, Fast Infoset, and GZIP Combinations

The parsing times for every combination are shown here in Figures 21 and 22 with GZIP so efficiency it is hardly visible. Again, as one would expect, these charts show that larger files naturally incur a high computing cost in parse time.

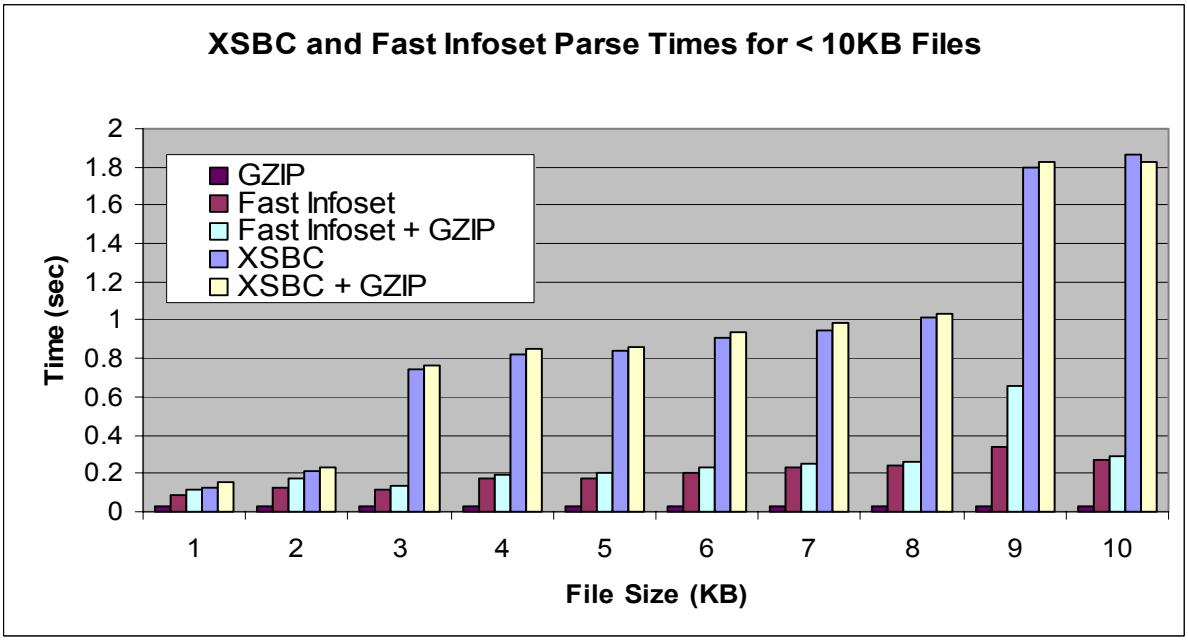

Figure 22. Parse Times for XSBC, Fast Infoset and GZIP Combinations for Files Less than 10 KB

F. ANALYSIS

1. Compression Performance

Examining Figures 16 and 17 reveal that a difference in compression is discernable between XSBC and Fast Infoset, but once GZIP is applied that difference is minimized. This is true for files larger than 10 KB. For files smaller than 10 KB, XSBC with GZIP clearly produces the greatest compression as seen in figures 17 and 19.

Without GZIP, XSBC provides better compression than Fast Infoset on files as large as 500 KB. Interestingly, Figure 18 shows that this file size is also provided the best compression resulting in a file 6% of its original size or a 96% bandwidth gain. Figure 19 reveals that when compression is applied to files 3-5 KB) XSBC with GZIP achieves 10-15% more compression than Fast Infoset. These are the sizes of the sample LINK16 XML files received from NC3A (see Appendix B). If file size is one's primary concern then XSBC with GZIP provides the best option especially for compressing files less than half of one megabyte.

2. Parse Time

The compression time for GZIP for a 3.5 MB sized file is 300 ms which is a fraction of the 18 seconds taken by XSBC and even the 6 seconds of Fast Infoset. GZIP is undeniably optimized for speed, making it the best solution for those seeking to reduce processing time on the server. However, the cost is counted in the compression as the file resulting from GZIP only is always 2% larger when not coupled with XSBC.

From Figure 21 it is clear to see that Fast Infoset's strength is its ability to parse larger files more quickly than XSBC. XSBC slows down more than Fast Infoset when parsing files larger than 300 KB. Between the two implementations, Fast Infoset is the solution of choice for files 1.5 MB or larger. However, using Fast Infoset over XSBC for files 300 KB and larger is largely dependent upon whether parsing speed or compression takes priority.

G. SUMMARY

For comparison among three algorithms, this chapter showed that XSBC coupled with GZIP provides superior compression performance on files smaller than 500 KB. However, it also showed that Fast Infoset maintains more consistent parsing times while parsing times for the unoptimized XSBC code begin to grow rapidly with file size.

VII. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

XML is a technology that, with its current track record, will continue to evolve and play a vital role in the future of data processing and transfer systems. As shown from the Binary XML Workshop a binary XML solution is needed. XML has become synonymous with web services and the large memory footprint that accompanies XML is a tremendous hindrance to integrating efficient web services. Not only is efficiency needed to enhance current services, but it could open the door for the widespread use of services by mobile devices.

The need for binary XML is valid as it covers a range of use cases from civilian to military and from data transfer to 3D rendering. A solution is definitely needed. The Binary XML Characterization Working Group determined that a solution that meets the foundational needs of the uses cases is possible. This solution will not be the silver bullet to solve all networking and memory issues, but provides a workable effective solution for most use cases. As the W3C is beginning to form the EXI working group to determine a standard, the US DoD and NATO should take a considerable interest in being involved with the process of choosing and testing candidates. With DoD and NATO supporting massive enterprise applications, a solution with tradeoffs that favor the military use case would provide enormous operational dividends.

In the meanwhile, it was shown that little additional work is needed for XSBC to integrate into NATO's TDL web service module NIRIS. As shown by the compression ratios, XSBC with GZIP provides the best performance between XSBC and Fast Infoset with files smaller than 500 kb. While XSBC lags slightly in parsing times for files smaller than 500 kb, the additional compression over Fast Infoset is worthwhile as computing power is plentiful for NIRIS and the data is not time critical down to a fraction of a second.

However, if N3CA is seeking an enterprise wide binary XML solution, the consistent compression of Fast Infoset and reduced parsing time as compared to XSBC makes it a more viable solution. This is especially true in dealing with the transfer of

larger files. In addition, Fast Infoset is better suited to a wider range of uses as it does not rely on a schema. This allows the encoding of an XML document without prior knowledge of its structure and content.

B. RECOMMENDATIONS FOR FUTURE WORK

Enabling XSBC to handle multiple namespace references would allow full integration into the NIRIS architecture and be a great benefit to NC3A. In addition, XSBC could be optimized for the specific link data based on the types that are to be compressed. An implementation needs to know, for instance, the number of significant digits for the required granularity. With this knowledge, XSBC can be augmented to reduce size of the data, before overall compression, by employing geometric compression techniques on that data.

XSBC currently can compress any well-formed XML document accompanied by a schema. Some XML related technologies however, contain capabilities that XML Schema is not sufficiently rich to model. One such example is XLST a language used to transform XML documents into other forms of XML. Using the XML Schema specification from W3C, one cannot write a schema that achieves 100% compliance with the XLST language. However, a pseudo-schema could be created that models XLST well enough to allow satisfactory XSBC compression. This technique can be applied to any other XML technologies in order to take advantage of the benefits offered by XSBC. Additional applications might include using a schema to encode and compress other schemas. This recursive technique can be used to compress large schema documents like the W3C Schema and the X3D schema.

Future work could also include the examination of possible civil uses of binary XML. The FAA could investigate the advantages of the air traffic control systems utilizing XML to provide a more comprehensive view of the air space similar to a military COP. In addition, in the event of an emergency where voice communications are non-functioning, the air traffic controllers could send vectoring information to the aircraft in a binary XML format to easily be interpreted and displayed for the pilots. Other possible civilian use might include integrating communications of emergency response

organizations. This means interoperability on the local and state level. As evidenced by the recent hurricane Katrina and Rita recovery efforts, having the fire and police departments sharing information along with ambulance drivers and medical helicopter pilots could lead to improvements in situational awareness and promote overall efficiency. On a larger level, during times of crisis (e.g. terrorist attacks, natural disasters, etc) local city authorities need a means of rapid communication and coordination with state officials. State authorities seeking to manage FEMA, Red Cross, and National Guard assets would also benefit from a fluid communication mechanism to ensure that time sensitive needs are not overlooked. Binary XML will also aid in other command and control functions.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. SOURCE CODE AVAILABILITY

XSBC is open source meaning the Java code is currently available on the Internet for download and use without financial restrictions. It is found at Sourceforge.net (http://cvs.sourceforge.net/viewcvs.py/xmsf/) which is a forum for open source development where code can be downloaded, altered, and then uploaded again by authorized users. Anonymous download of this source code is supported and can be used to obtain XSBC.

Sourceforge.net uses an application called Concurrent Versioning System (CVS). This technology allows for multiple developers to be using and improving code simultaneously. CVS then collaborates and de-conflicts changes so that all the added functionally is present. This prevents programmers from overwriting or hindering one another's work.

CVS capability is integrated into netBeans, an open source Java IDE, and can be invoked from netBeans, or from a simple open source application like WinCvs, which can be downloaded for free at http://www.wincvs.org. Once WinCvs is installed, instruction for anonymous download of XSBC can be found here at http://sourceforge.net/cvs/?group_id=86243. One only needs to replace "*modulename"* with "XSBC" in the example commands and then execute them. In order to be involved with the development process and be able to add functionality and upload changes, please contact the MOVES Institute.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. EXPLANATION OF SIMPLE EXPERIMENT AND SOURCE CODE

A. MOTIVATION

The SimpleMain program began as a way to duplicate the tests performed at NC3A in which LINK16 J-series messages were serialized via XSBC, then sent across the network using NIRIS and then deserialized on the other side. Due to time constraints, the focus of the work was on successfully integrating XSBC into NIRIS and an effective means of capturing basic network performance metrics of XSBC was (rightfully so) a lower priority. As a result, the author and Terry Norbraten sought to reproduce a simplified version of the NC3A test in order to serve as a live demo tool, as well as a way to provide a testbed to see if XSBC can meet specified network requirements. This tool is ideally suited for testing to see if XSBC can effectively serialize *X* number of files and send them across the network of *Y* bandwidth in *Z* number of seconds.

B. DESCRIPTION

The SimpleMain program consists of three Java files: SimpleMain, SimpleServer, and SimpleClient.

1. SimpleMain

SimpleMain just provides a main class from which to launch either the server or client. If the argument 'server' is applied when running SimpleMain then the SimpleServer is run and begins listening on port 4040. The SimpleClient is run by applying the arguments 'client' and 'x.x.x.x' where the latter is the IP address of the server. If no IP address is supplied then a default loopback address is used.

2. SimpleClient

SimpleClient, when instantiated, stores all the files in the specified directory (currently hard coded as C:/xsbc/examples/NC3ABinXml) into a file array. Then a for-loop takes each file and creates an instance of XsbcSerializer which makes calls to

serialize the file and stream it via Transmission Control Protocol (TCP) across the network. XsbcSerializer is incorporated in the standard XSBC jar file, but we modified it so it injects a timestamp directly after opening the output stream as well as using time stamps to measure the time to serialize the file. SimpleClient then uses a 'get' method we added to capture the serialization times for each file. It then creates a text file containing each flile's serialization time as a new line for easy portability into a spreadsheet for analysis.

SimpleClient creates a text file locally named "serial.txt" to record the results of the serialization times.

3. SimpleServer

SimpleServer, when instantiated, opens a socket on port 4040 and then enters a for-loop. Currently, the for-loop is hard coded for 960 iterations as that was the exact number of test files, but that is easily changed. For each iteration in the loop, an instance of an XsbcTransaction thread is created. This class is also found in the XSBC jar, but again we have modified it. It simply receives a TCP stream of compressed XSBC documents and de-serializes them back into *.xml files. Our modifications include the addition of a cubbyhole class and timestamps to measure latency and Deserialization time. The first eight bytes of each file received from XsbcSerializer contains the timestamp we inserted into the stream. This is stripped out and compared with the current time to calculate latency. Deserialization time is calculated by taking timestamps before and after the deserialization method is called.

The cubbyhole class is vital in extracting the information from each file as it synchronizes with the threads and extracts the information before the threads terminate. Before implementing the cubbyhole technique, the SimpleServer was unable to access any of the timestamps to calculate latency as the threads within it spawned and died before it could extract the information. The code was modified from Sun's open source tutorial (http://java.sun.com/docs/books/tutorial/essential/threads/).

SimpleServer creates two files locally named "deserial.txt" and "latency.txt" to record their respective metrics.

C. DATA SAMPLES

Figures 19 and 20 show the latency data from two sample runs of the SimpleMain: one with two machines connected into the same hub on a wired LAN and the other with the two same machines connecting wirelessly into the LAN. The sample data was collected by serializing 960 J-series LINK16 messages received from NC3A. Each file was 3-5kb in size. The latency data was collected from the "latency.txt" files created after each run and imported in a spreadsheet for display.

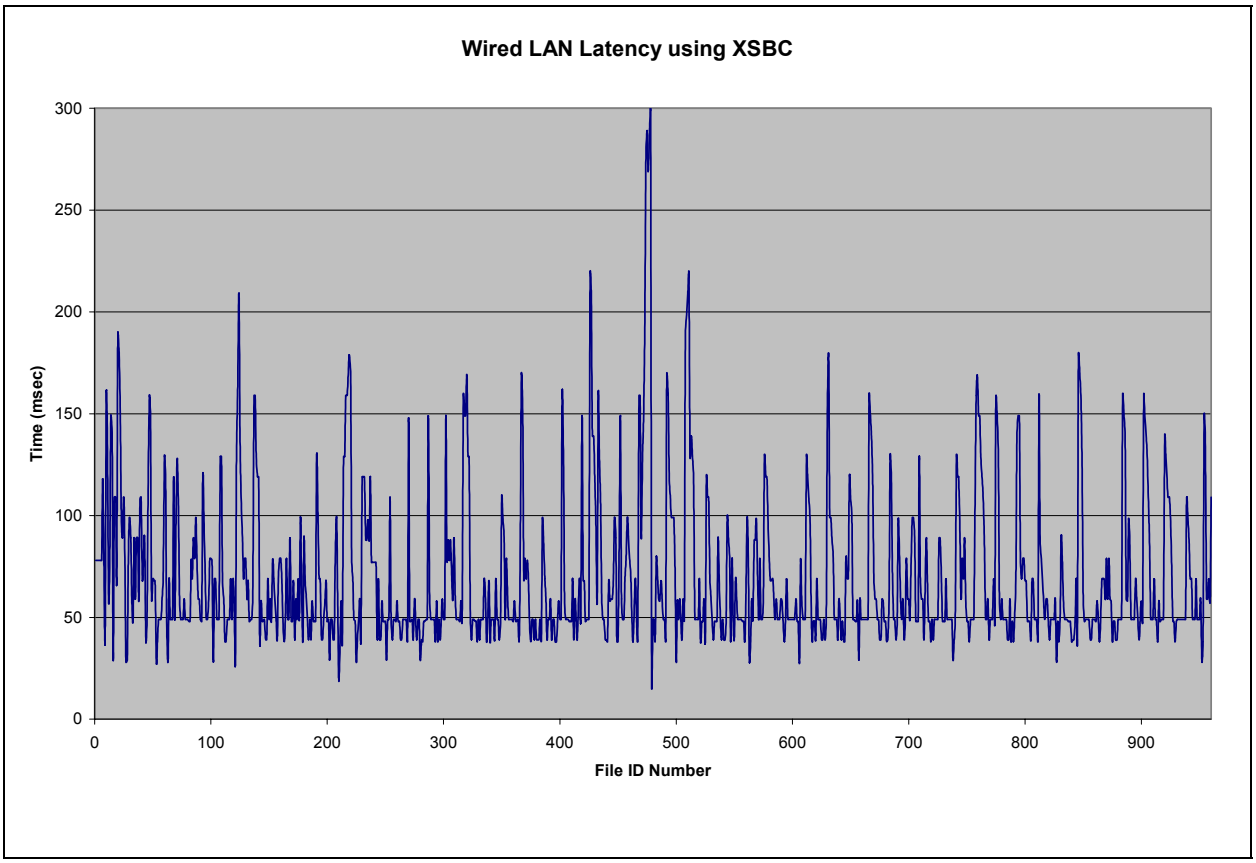

Figure 23. Network latency of wired LAN using XSBC

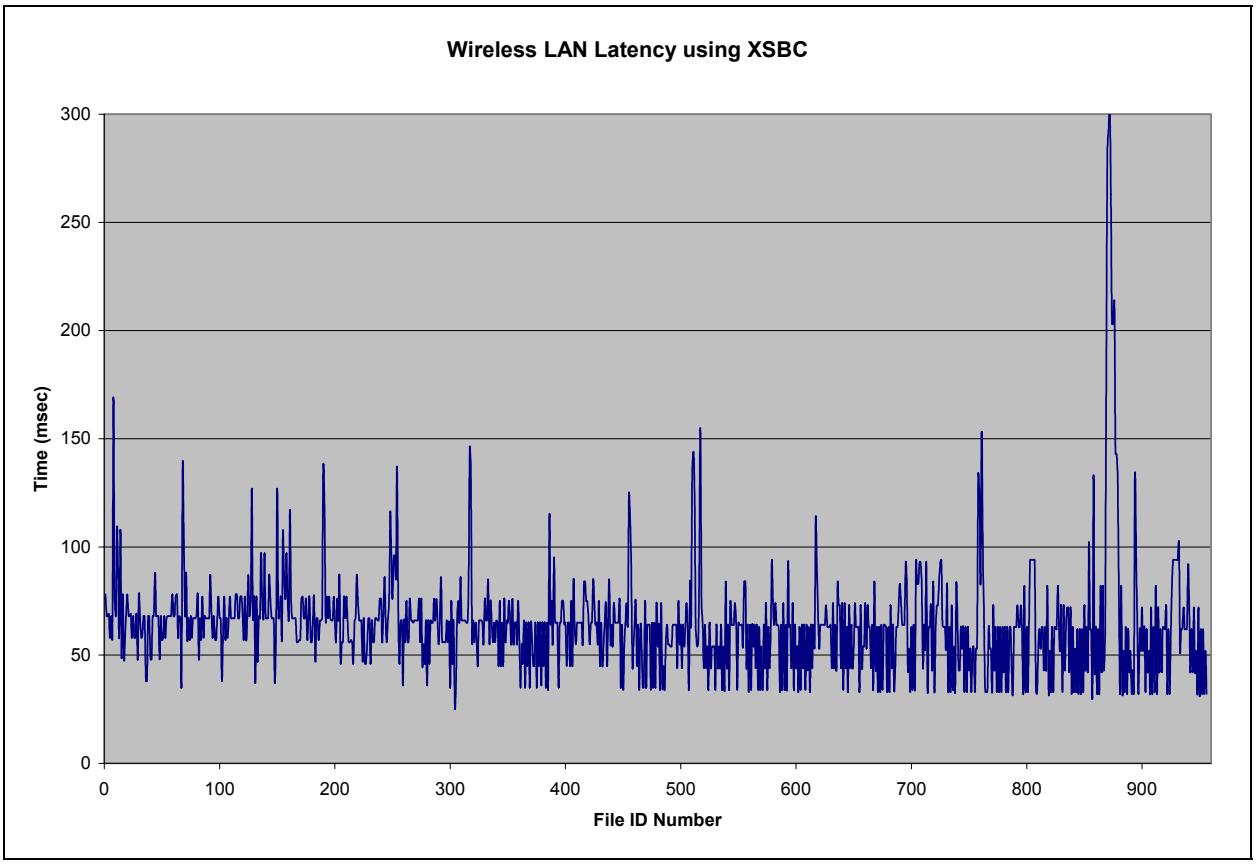

Figure 24. Network latency of a wireless connection to the LAN using XSBC

D. SOURCE CODE

1. SimpleMain.java

```
/*
 * SimpleMain.java
 *
 * Created on August 24, 2005, 4:25 PM
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 *
 * Description: This class serves as a common launching point for both the
 * SimpleClient and SimpleServer via command line.  If SimpleMain is run with
 * the argument 'server' then SimpleServer begins listening on port 4040.  If
 * SimpleMain is run with the argument 'client' it begins the client on
 * 4040 with the default address being a loopback.  However one can specify
 * an IP address as a second argument.
 */

package matt.bayer.xsbc.project;

import cubbyhole.CubbyHole;

/**
 *
 * @author mebayer
 */
public class SimpleMain {

    public CubbyHole c;
    /** Creates a new instance of SimpleMain */
    public SimpleMain() {

        c = new CubbyHole();
        SimpleServer ss = new SimpleServer(c);
        SimpleClient sc = new SimpleClient("127.0.0.1");
        ss.start();
        sc.main();

    }


    public SimpleMain(String args, String address) {

        if (args.equals("server"))
        {
            c = new CubbyHole();
            SimpleServer ss = new SimpleServer(c);
            ss.start();
        }
        else if (args.equals("client"))
        {
            SimpleClient sc = new SimpleClient(address);
            sc.main();
        }


    } // end SimpleMain(String args)
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        if (args.length == 0){
            new SimpleMain();
        } else if (args[0].equals("server")){
            new SimpleMain(args[0], "");
        } else if (args[0].equals("client")){
            new SimpleMain(args[0], args[1]);
        } else{
            System.out.println("Usage: specify as server or client");
        }
    }

}
```

2. SimpleClient.java

```
/* Program:
 *
 * Author:       Matt Bayer, NPS
 * Modifier:
 *
 * Created on:   August 19, 2005, 12:46 PM
 * Modified on:
 *
 * File:         SimpleClient.java
 *
 * Compiler:     netBeans IDE 4.1 (External), j2sdk1.4.2_08
 * O/S:          Windows XP Home Ed. (SP2)
 *
 * Description:
 *
 * Information:  Using xsbc-0.92 created by Alan Hudson of Yumetech
 *               Inc.  An Java open source API.
 */

/*****************************************************************************
 *                        Web3d.org Copyright (c) 2004
 *                               Java Source
 *
 * This source is licensed under the GNU LGPL v2.1
 * Please read http://www.gnu.org/copyleft/lgpl.html for more information
 *
 * This software comes with the standard NO WARRANTY disclaimer for any
 * purpose. Use it at your own risk. If there's a problem you get to fix it.
 *
 *****************************************************************************/

package matt.bayer.xsbc.project;

import java.io.File;
import java.io.FileWriter;
import org.web3d.xmsf.xsbc.stream.XsbcSerializer;
```

```
/**
 *
 * @author <a href="mailto:mebayer@nps.edu">Matt Bayer, NPS</a>
 *
 * @see SimpleClient
 */
public class SimpleClient {
//public class SimpleClient extends Thread{

   /* DATA MEMBERS */

   /* Private */

   /* Public */

   /** GZipped or not */
   public static boolean isZipped = false;

   /** Designated port */
   public static final int PORT = 4040;

   /** Instance of our working files */
   public static File file;

   /** IP Hostname */
   public static String host = "128.0.0.1";

   /** Substring pointer to our test .xml file */
   public static final String TEST_FILES = "/j35/";

   /** String pointer to NC3A message directory */
   public static final String USER_DIR = "C:/xsbc/examples/NC3ABinXml";

   /** Instance of our XSBC server */
   public static XsbcSerializer s;
   private static FileWriter fwserial;

   public SimpleClient(String host)
   {
       this.host = host;
   }

   /* MAIN METHOD */

   /**
    * Entry point for the program
    *
    * @param args the command line entry arguments if any
    */
   public static void main(){
   //public void run(){

   try{
   fwserial = new FileWriter("c:/xsbc/data/serial.txt");

   File file = new File(USER_DIR + TEST_FILES);
   File[] fArray = file.listFiles();

   for (int i = 0; i < fArray.length; i++) {
       s = new XsbcSerializer(fArray[i].getPath());
       s.writeNetwork(host, PORT, isZipped);

       fwserial.write(String.valueOf(s.getSerialTime()));
```

```
        fwserial.write("\n");
        fwserial.flush();

   }
        fwserial.close();
   } // end try
   catch (Exception e)
   { e.printStackTrace();}


   } // end main()

} // end class file SimpleServer.java
```

3. SimpleServer.java

```
/*
 * SimpleServer.java
 *
 * Created on August 19, 2005, 12:46 PM
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */

package matt.bayer.xsbc.project;

import java.io.File;
import java.io.FileWriter;
import java.io.InputStream;
import java.net.*;
import org.web3d.xmsf.xsbc.stream.XsbcTransaction;
import cubbyhole.CubbyHole;


/**
 *
 * @author <a href="mailto:mebayer@nps.edu">Matt Bayer, NPS</a>
 *
 * @see SimpleServer
 */
public class SimpleServer extends Thread{

   /** Specify working port */
   public static final int PORT = 4040;

   /** Pointer to our required schema */
   public static final File SCHEMA =
           new File("C:/xsbc/examples/NC3ABinXml/jmsg_input.xsd");

   /** Instance of our TCP socket */
   public static ServerSocket socket;

   /** Instance our XSBC client */
   public static XsbcTransaction trans;

   /** Instance of a thread for the client */
   public static Thread t;
   private static FileWriter fwdeserial;
```

```
   private static FileWriter fwlatency;
   private CubbyHole ch;

   public SimpleServer(CubbyHole c)
   {
       // associates passed parameter c CubbyHole with local ch CubbyHole
       ch = c;
   }

   /**
    * Entry point for the program
    *
    * @param args the command line entry arguments if any
    */
   public void run(){

      try {

         socket = new ServerSocket(PORT);
         fwdeserial = new FileWriter("c:/xsbc/data/deserial.txt");
         fwlatency = new FileWriter("c:/xsbc/data/latency.txt");


         for (int i = 0; i < 960; i++) {

            trans = new XsbcTransaction(socket.accept(), i, SCHEMA, ch);
            t = new Thread( trans, "xsbcTrans" );
            t.start();


            fwlatency.write(String.valueOf(ch.getLatency()));
            fwdeserial.write(String.valueOf(ch.getDeserial()));
            // fwdeserial.write(String.valueOf(trans.getDeserialTime()));
            // fwlatency.write(String.valueOf(trans.getLatencyTime()));

            fwdeserial.write("\n");
            fwdeserial.flush();

            fwlatency.write("\n");
            fwlatency.flush();

         } // end for loop
         fwlatency.close();
         fwdeserial.close();

      } catch ( Exception e ) {

         e.printStackTrace();

      } // end try-catch block

   } // end main

} // end class file SimpleServer.java
```

4. XsbcSerializer.java

```
/*
Copyright (c) 1995-2005 held by the author(s).  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer
      in the documentation and/or other materials provided with the
      distribution.
    * Neither the names of the Naval Postgraduate School (NPS)
      Modeling Virtual Environments and Simulation (MOVES) Institute
      (http://www.nps.edu and http://www.MovesInstitute.org)
      nor the names of its contributors may be used to endorse or
      promote products derived from this software without specific
      prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
*/

/* Program:     Extensible Markup Language (XML) Schema-based Binary
 *              Compression (XSBC) with Forward Error Correction (FEC)
 *
 * Author:      Duane Davis
 * Modifier:    Terry Norbraten, Matt Bayer NPS MOVES
 *
 * Created on:  Janurary 31, 2004
 * Modified on: August 20, 2005
 * Time:        1415:18
 *
 * File:        XsbcSerializer.java
 *
 * Compiler:    netBeans IDE 4.1 (External), j2sdk1.4.2_08
 * O/S:         Windows XP Home Ed. (SP2)
 *
 * Description: Reads and XML document and compresses it with XSBC
 *
 * Information: Modified to feed timestamp into OutputStream for network
 *              latency metric analysis, calculate serialization time and move
 *              setCompressedMethod() to the constructor
 */
package org.web3d.xmsf.xsbc.stream;

// Standard library imports
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
```

```
import java.io.IOException;
import java.io.OutputStream;
import java.io.FileWriter;

import java.net.InetAddress;
import java.net.Socket;

import java.util.Date;
import java.util.zip.GZIPOutputStream;

// Application specific local imports
import org.web3d.xmsf.xsbc.DocumentWriter;
import org.web3d.xmsf.xsbc.datatypes.SimpleType;
import org.web3d.xmsf.xsbc.util.DecodeEncodePrimitives;
import org.web3d.xmsf.xsbc.util.Utilities;

/**
 * Reads an XML file and compresses it using XSBC.  The result can be written to
 * a file or the network. </p>
 *
 * @version $Revision: 1.6 $
 * <p>
 *   <dt><b>Latest Modifications:</b>
 *   <pre><b>
 *     Date:     01 AUG 2005
 *     Time:     1318:10
 *     Author:   <a href="mailto:tdnorbra@nps.edu?subject=xsbc">Terry Norbraten</a>
 *     Comments: Modified to optimize Output Stream wrapping for serialization
 *   </b></pre>
 *   <pre><b>
 *     Date:     August 20, 2005
 *     Time:     1415:18
 *     Author:   <a href="mailto:tdnorbra@nps.edu?subject=xsbc">Terry Norbraten</a>
 *     Comments: Modified to feed timestamp into OutputStream for network
 *               latency metric analysis, calculate serialization time and move
 *               setCompressedMethod() to the constructor
 *   </b></pre>
 * </p>
 * @author <a href="mailto:dtdavis@nps.edu">Duane T. Davis</a>
 *<P>
 *Contact: Don Brutzman (<A
HREF="http://web.nps.navy.mil/~brutzman"><i>web.nps.navy.mil/~brutzman</i></A>)
 * <A HREF="mailto:brutzman@nps.navy.mil(Don Brutzman)?subject=xsbc feedback
"><i>brutzman@nps.navy.mil</i></A>
 *<P>
 *@see XsbcTransaction
 */
public class XsbcSerializer {

   /* DATA MEMBERS(s) */

   /** Writer that will facilitate XSBC serialization */
   private DocumentWriter writer;

   /** Input, Output files */
   private File inFile, outFile;
   private long serialTime;
```

```
/* CONSTRUCTORS(s) */

/**
 * Creates a new instance of XsbcSerializer
 *
 * @param inFilePath path (including filename) of input XML file
 */
public XsbcSerializer(String inFilePath) {

   try {

      inFile = new File(inFilePath);
      writer = new DocumentWriter(inFile.getPath());
      SimpleType.setCompressionMethod(
              SimpleType.COMPRESSION_METHOD_SMALLEST_NONLOSSY );
      serialTime = 0;

   } catch(Exception e) {
      e.printStackTrace();

   } // end try-catch block

} // end constructor

/* PUBLIC METHODS(s) */

/**
 * Writes the Xsbc data to a specified file
 *
 * @return indication of success or failure of writing XSBC
 * @param outFilePath Path (including name) of the output file
 * @param zipped true if output file is to be compressed (zipped)
 */
public int writeFile( String outFilePath, boolean zipped ) {

   try {

      int result;
      setOutFile( new File( outFilePath ) );
      FileOutputStream fos = new FileOutputStream( getOutFile() );

      if ( zipped )
         result = writeZippedStream( fos );
      else
         result = writeStream( fos );

      return result;

   } catch ( IOException ioe ) {

      ioe.printStackTrace();
      return( 0 );

   } // end try-catch block

} // end writeFile()
```

```
   /**
    * Writes the XSBC data to a network address by opening a socket
    * Closes the socket once write is complete
    *
    * @param host the server Hostname
    * @param port the port on host to connect to
    * @param zipped is true if the file is to be compressed, false if not
    * @return an indication of success or failure
    */
   public int writeNetwork( String host, int port, boolean zipped ) {

      int result = 0;
      Socket socket = null;
      long timeStamp = 0, timeNow = 0;

      try {

         socket = new Socket( InetAddress.getByName( host), port );
         OutputStream os = socket.getOutputStream();

         // Insert a time stamp into output stream for network latency metric
         // analysis
         os.write( DecodeEncodePrimitives.encodeLong(new Date().getTime()) );
         os.flush();

         timeNow = new Date().getTime();
         timeStamp = timeNow;

         if (zipped)
            result = writeZippedStream( os );
         else
            result = writeStream( os );

         timeNow = new Date().getTime();

      } catch (Exception e) {

         Utilities.debugOut("Unable to establish network connection for " +
                 "XSBC-compressed archive transfer");
         Utilities.debugOut(e.getMessage());
//         e.printStackTrace();

      } // end try-catch block

      try {

         if(socket != null)
            socket.close();

      } catch (IOException e) {}

      Utilities.traceOut("Serializer processing time for " + inFile + " was: "
              + String.valueOf(timeNow - timeStamp ) + " ms");
      serialTime = timeNow - timeStamp;

      return result;

   } // end writeNetwork()
```

```
   /* GETTER(s) and SETTER(s) */

   /**
    * Sets the out going *.xml file for encoding (tdn)
    *
    * @param file the *.xml file to encode
    */
   public void setOutFile( File file ) {

      outFile = file;

   } // end setOutFile()

   /**
    * Retrieves the out going *.xml file for encoding (tdn)
    *
    * @return the *.xml for encoding
    */
   public File getOutFile() {

      return outFile;

   } // end getOutFile()

   /* PROTECTED METHOD(s) */

   /* PRIVATE METHOD(s) */

   /**
    * Writes the XSBC data to an output stream.  Does not open stream.</p>
    *
    * @param stream the OutputStream to write Xsbc to
    * @return the indication of success or failure
    */
   private int writeStream( OutputStream stream ) {

      try {

         DataOutputStream dos = new DataOutputStream( stream );
         writer.serialize( dos );
         dos.close();
         stream.close();
         return(1);

      } catch (Exception e) {
         Utilities.debugOut("Unable to write XSBC to stream\n" + e.getMessage());
//         e.printStackTrace();
         return(0);
      } // catch (Exception e)

   } // end writeStream()

   /**
    * Writes the XSBC data to an output stream in zipped form.  Does not open
    * the stream. </p>
    *
    * @param stream the output stream to write compressed Xsbc to
    * @return the indication of success or failure in serializing the xml file
    * @exception IOException
    *            If there was a problem in XSBC serializing the input file
    */
   private int writeZippedStream( OutputStream stream ) {
```

```
      try {

         GZIPOutputStream zipStream = new GZIPOutputStream( stream );
         DataOutputStream dos = new DataOutputStream( zipStream );
         writer.serialize( dos );
         dos.close();
         zipStream.close();
         stream.close();
         return( 1 );

      } catch ( IOException ioe ) {

         Utilities.debugOut( "Unable to write XSBC to Zip Stream" );
         ioe.printStackTrace();
         return( 0 );

      } // end try-catch block

   } // end writeStreamZipped()

   public long getSerialTime()
   {
        return serialTime;
   }

} // end class file XSBCSerializer.java
```

5. XsbcTransaction.java

```
/*
Copyright (c) 1995-2005 held by the author(s).  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer
      in the documentation and/or other materials provided with the
      distribution.
    * Neither the names of the Naval Postgraduate School (NPS)
      Modeling Virtual Environments and Simulation (MOVES) Institute
      (http://www.nps.edu and http://www.MovesInstitute.org)
      nor the names of its contributors may be used to endorse or
      promote products derived from this software without specific
      prior written permission.
```

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
*/

/* Program:     Extensible Markup Language (XML) Schema-based Binary
 *              Compression (XSBC) with Forward Error Correction (FEC)
 *
 * Author:      Duane Davis
 * Modifier:    Terry Norbraten, NPS MOVES
 *
 * Created on:  February 01, 2004: 0006
 * Modified on: August 20, 2005
 * Time:        1415:18
 *
 * File:        XsbcTransaction.java
 *
 * Compiler:    netBeans IDE 4.1 (External), j2sdk1.4.2_08
 * O/S:         Windows XP Home Ed. (SP2)
 *
 * Description: Accepts a TCP stream of compressed data to de-serialize and/or
 *              uncompresses and de-serialize back into *.xml form
 *
 * Information: Modified to read timestamp from InputStream for network latency
 *              metrics analysis and to calculate deserialization  time
 */
package org.web3d.xmsf.xsbc.stream;

// Standard library imports
import java.io.BufferedInputStream;
import java.io.FileOutputStream;
import java.io.File;
import java.io.InputStream;
import java.io.FileWriter;
import java.net.Socket;
import java.util.Date;

// Application specific local imports
import org.apache.batik.util.ParsedURLData;
import org.web3d.xmsf.xsbc.*;
import org.web3d.xmsf.xsbc.datatypes.*;
import org.web3d.xmsf.xsbc.util.DecodeEncodePrimitives;
import org.web3d.xmsf.xsbc.util.Utilities;
import cubbyhole.CubbyHole;
```

```
/**
 * Accepts a TCP stream of compressed data to de-serialize and/or uncompresses
 * and de-serialize back into *.xml form.
 *
 * @version $Revision: 1.3 $
 * <p>
 *   <dt><b>Latest Modification:</b>
 *   <pre><b>
 *     Date:     08 MAY 2005
 *     Time:     2041:20
 *     Author:   <a href="mailto:tdnorbra@nps.edu?subject=xsbc">Terry Norbraten</a>
 *     Comments: Modified to accept a schema for *.xml deserialization  and to
 *               handle a non-GZipped InputStream.
 *   </b></pre>
 *   <pre><b>
 *     Date:     August 20, 2005
 *     Time:     1415:18
 *     Author:   <a href="mailto:tdnorbra@nps.edu?subject=xsbc">Terry Norbraten</a>
 *     Comments: Modified to read timestamp from InputStream for network latency
 *               metrics analysis and to calculate deserialization  time
 *   </b></pre>
 * </p>
 * @author <a href="mailto:dtdavis@nps.edu">Duane T. Davis</a>
 * <P>
 *Contact: Don Brutzman (<A
HREF="http://web.nps.navy.mil/~brutzman"><i>web.nps.navy.mil/~brutzman</i></A>)
 * <A HREF="mailto:brutzman@nps.navy.mil(Don Brutzman)?subject=xsbc feedback
"><i>brutzman@nps.navy.mil</i></A>
 *<P>
 *@see XsbcSerializer
 */
public class XsbcTransaction implements Runnable {

   /* DATA MEMBER(s) */

   /* Public */

   /* Protected */

   /* Private */

   /** Required stream for the XSBCReader */
   private BlockDataInputStream bdis;

   /** Required stream for the BlockDataInputStream */
   private BufferedInputStream bis;

   /** Final file  and schema objects */
   private File outFile,
                schemaLoc;

   /** Required stream for the XMLWriter */
   private FileOutputStream fos;

   /** The socket's TCP input stream */
   private InputStream iStream;

   /** Our TCP socket */
   private Socket socket;

   /** Used to hold schema info for XML document deserialization  */
   private TableManager tableManager;
```

```
/** Writer for the deserialized XML file */
private XMLWriter writer;

/** Reader for XSBC to XML deserialization  */
private XSBCReader reader;

/** Dummy id tag for the mission results file */
private int transactionId;

/** Metrics for various processing times */
private long deserializationTimeBefore,
             deserializationTimeAfter;

/** Point to the data cache folder */
private final String DATACACHE = System.getProperty( "user.dir" ) +
        "/dataweb/results/";

/** File base name for resulting .xsbc file */
private final String CACHEFILE = "resultXsbc";

private CubbyHole ch;
/* CONSTRUCTOR(s) */

/**
 * Creates a new instance of XsbcTransaction
 *
 * @param socket the socket over which the transaction will take place
 * @param id the of this transaction
 * @param schema the schema to load into the XSBC Table Manager
 */
public XsbcTransaction( Socket socket, int id, File schema ) {

       this.socket = socket;
       transactionId = id;
       schemaLoc = schema;

} // end constructor

public XsbcTransaction(Socket socket, int id, File schema, CubbyHole c)
{
    this.socket = socket;
    transactionId = id;
    schemaLoc = schema;

    // CubbyHole from project main is passed to local CubbyHole
    ch = c;
}

/* THREAD PROCESS */

/** Processes the XSBC transaction */
public void run() {

  Utilities.traceOut( "External network connection made.  Begin XSBC " +
           "file upload" );

   SimpleType.setCompressionMethod(
              SimpleType.COMPRESSION_METHOD_SMALLEST_NONLOSSY );

   try {

      iStream = this.socket.getInputStream();
```

```
         // Extract timestamp sent from XsbcSerializer.  Long integers are eight
         // bytes in length.
         byte[] b = new byte[8];
         iStream.read(b, 0, b.length);
         long timeStamp = DecodeEncodePrimitives.decodeLong(b);
         long timeNow = new Date().getTime();
         Utilities.traceOut("Network latency is calculated at: " +
                 String.valueOf(timeNow - timeStamp) + " ms");

         ch.putLatency(timeNow - timeStamp);


         // Check for iStream in GZip format
         bis = new BufferedInputStream( ParsedURLData.checkGZIP( iStream ) );
        Utilities.traceOut( "Loading XSBC data .... " );
        Utilities.traceOut( "XSBC using schema located at " +
schemaLoc.toURL() );
         bdis = new BlockDataInputStream( bis );
         outFile = new File( DATACACHE + CACHEFILE + transactionId + ".xml" );
         fos = new FileOutputStream( outFile );
         writer = new XMLWriter( fos );
         tableManager = new TableManager( schemaLoc.toURL() );
         reader = new XSBCReader( tableManager );

         deserializationTimeBefore = new Date().getTime();
         reader.read( bdis, writer );
         deserializationTimeAfter = new Date().getTime();

         // Time stamp created for NC3A metrics
         Utilities.traceOut( "Deserialization  processing time for " + outFile
                 + " was: "
                 + ( deserializationTimeAfter - deserializationTimeBefore )
                 + " ms" );

         ch.putDeserial(deserializationTimeAfter - deserializationTimeBefore);

         // Reclaim resources
         iStream.close();
         bis.close();
         bdis.close();
         fos.close();
         Utilities.traceOut( "Archive file successfully uploaded" );

         if ( socket != null ) {

            socket.close();
            Utilities.traceOut( "Transaction socket now closed" );

         } // end if

      } catch ( Exception e ) {

            System.err.println( "Problem encountered while de-serializing " +
                    "file" );
            e.printStackTrace();

      } // end try-catch block

   } // end run()


} // end class file XsbcTransaction.java
```

6. CubbyHole.java

```
/*
 * CubbyHole.java
 *
 * Created on August 24, 2005, 3:15 PM
 */

package matt.bayer.xsbc.project;


/**
 *
 * @author mebayer modified from
 * http://java.sun.com/docs/books/tutorial/essential/threads/
 */
public class CubbyHole {
    private long lcontents, dcontents;
    private boolean available = false;

    public synchronized long getLatency() {
        while (available == false) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        available = false;

        notifyAll();
        return lcontents;
    }

    public synchronized long getDeserial() {
        while (available == false) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        available = false;

        notifyAll();
        return dcontents;
    }

    public synchronized void putLatency(long value) {
        while (available == true) {
            try {
                wait();
            } catch (InterruptedException e) { }
        }
        lcontents = value;
        available = true;

        notifyAll();
    }

    public synchronized void putDeserial(long value) {
        while (available == true) {
            try {
                wait();
            } catch (InterruptedException e) { }
```

```
        }
        dcontents = value;
        available = true;

        notifyAll();
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Air Land Sea Application Center. (2000). *TADIL J: Introduction to Tactical Digital Information Link J and Quick Reference Guide.* Retrieved October 3, 2005 from https://www.doctrine.usmc.mil/signpubs/r325c.pdf

Alberts, D. S., Garstka, J. J., & Stein, F. P. (1999). *Network Centric Warfare: Developing and Leveraging Information Superiority* (2nd ed.). Washington D.C.: CCRP.

Blais, C., Brutzman, D., Drake, D., Moen, D., Morse, K., Pullen, M., and Tolk, *A., Extensible Modeling and Simulation Framework (XMSF) 2004 Project Summary Report*, NPS-MV-05-002, Naval Postgraduate School MOVES Institute, Monterey, CA, 4 February 2005.

Boss, B. (2003). *XML in 10 points.* Retrieved September/20, 2005 from http://www.w3.org/XML/1999/XML-in-10-points

Bray, Tim., Paoli, Jean., Sperberg-Mcqueen., Maler, Eve. & Yergeau, Francois. (2004) *Extensible Markup Language (XML) 1.0 (Third Edition)* Retrieved from September 30, 2005 from http://www.w3.org/TR/2004/REC-xml-20040204/

Cokus, M., & Percias-Geertsen, S. (2005a). *XML Binary Characterization Use Cases.* Retrieved September 14, 2005 from http://www.w3.org/TR/xbc-use-cases/

Cokus, M., & Percias-Geertsen, S. (2005b). *XML Binary Characterization Properties.* Retrieved September 14, 2005 from http://www.w3.org/TR/xbc-properties/

Conner, M., & Mendelsohn, N. (2003). *Issues Relating to the Creation of a Binary Interchange Standard for XML*. Armonk, NY: IBM Corporation. August 23, 2005.

Cown, John. & Tobin, Richard. (2004). *XML Information Set (Second Edition).* Retrieved September 30, 2005 from http://www.w3.org/TR/xml-infoset/

Downs, E. (2005). *Digital Datalinks.* Retrieved August 25, 2005 from http://www4.janes.com.libproxy.nps.navy.mil/K2/doc.jsp?t=Q&K2DocKey=/content1/janesdata/yb/jav/jav_9709.htm@current&QueryText=%3CAND%3E(%3COR%3E((%5B80%5D(TADIL+%3CAND%3E+J)+%3CIN%3E+body)%2C+(%5B100%5D(%5B100%5D(TADIL+%3CAND%3E+J)+%3CIN%3E+title)+%3CAND%3

Goldman, O., & Lenkov, D. (2005). *XML Binary Characterization.* Retrieved September 2, 2005 from http://www.w3.org/TR/xbc-characterization/

Gudgin, M. (2004). *Understanding Infosets.* Retrieved September 06, 2005 from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/xmlinfoset.asp

Howland, P. E., (2004). *Near Real-Time Tactical Data Services for Network Enabled Operations.* Unpublished manuscript. Retrieved April 13, 2005,

Hunter, D., Cagle, K., Dix, C., Kovack, R., Pinnock, J., & Rafter, J. (2003). Chapter 6: XML Schema. *Beginning XML* (2nd ed.) (pp. 217). Indianapolis: Wiley Publishing Inc.

Leung, T. W. (2004). Chapter 10: XML Security. *XML Development with Apache Tools Xerces, Xalan, FOP, Cocoon, Axis, Xindice* (pp. 455). Indianapolis: Wiley Publishing Inc.

Lilley, C. & Karmarkar, A.. (2003). Report From the W3C Workshop on Binary Interchange of XML Information Item Sets. Santa Clara, California, Retrieved October 3, 2005,

McGregor, D. (2000). *Dial-A-behavior Protocol.* Retrieved October 20, 2005 from http://www.movesinstitute.org/~mcgredo/dabp/

Muller, D. K. (2003). *NATO and XML.* Retrieved August 19, 2005 from http://www.gca.org/papers/xmleurope2000/papers/s16-03.html

Norbraten, T D. (2004) Utilization of Forward Error Correction (FEC) Techniques with Extensible Markup Language (XML) Schema-Based Binary Compression (XSBC) Technology. (Master's Thesis, Naval Postgraduate School). (Public Release)

Orchard, D. (2003). *Finding the 80/20 point for XML performance* (Position Presentation. San Jose, California: BEA Systems.

Pal, S., Marsh, J., & Layman, A. (2003). *A Case Against Standardizing Binary Representation of XML* (Position Paper. Redmond, WA: Microsoft Corporation.

Pike, J. (2000). *Tactical Data Information Links.* Retrieved August 24, 2005 from http://www.fas.org/irp/program/disseminate/tadil.htm

Raggett, D., Le Hors, A., & Jacobs, I. (1999) *HTML 4.01 Specification*. Retrieved October 3, 2005 from http://www.w3.org/TR/REC-html40/sgml/entities.html

Sandoz, P. (2005). *Fast Infoset and Fast Web Services.* Sun Microsystems. Retrieved August 7, 2005, from www.itu.int/ITU-T/studygroups/com17/tutorials/tutorial_2005_03_30_sandoz.pdf

Sandoz, P., & Percias-Geersten, S. (2005). *Fast Infoset @ Java.net.* Retrieved September 05, 2005 from http://idealliance.org/proceedings/xtech05/papers/04-01-01/

Sandoz P., Percias-Geersten, S & Brutzman, D. (2005) *Fast Infoset: A Fast, Effective and Practical Binary Encoding for the XML Infoset.* 2005 JavaOne Conference, Sun Microsystems.

Sandoz, P., Triglia, A. & Percias-Geertsen, S. (2004). *Fast Infoset.* Retrieved September 5, 2005 from http://java.sun.com/developer/technicalArticles/xml/fastinfoset/

Serin, E. (2003). Design and Test for the Cross Format Schema Protocol (XFSP) for Networked Virtual Environments. (Master's Thesis, Naval Postgraduate School)., (Public Release)

Sun Microsystems. (2005). *The Java Tutorial: Data types.* Retrieved September 20, 2005 from http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html W3C Recommendations. (2004).

Web3D Consortium. (2003). *X3D Compressed Binary Encoding Request for Proposals (RFP)*. Retrieved September 28, 2005 from http://www.web3d.org/x3d/binary/index.html

Web3D Consortium. (2005a). *X3D Overview*. Retrieved September 28, 2005 from http://www.web3d.org/x3d/overview.html

Web3D Consortium. (2005b). *Extensible 3D Encodings. Part 3 Binary Encodings: Introduction*. Retrieved September 28, 2005 from http://www.web3d.org/x3d/specifications/ISO-IEC-19776-3-CD-X3DEncodings-CompressedBinary/

Wikipedia: The Free Encyclopedia. (2005.) *Moore's law.* Retrieved September, 30 2005 from http://en.wikipedia.org/wiki/Moore%27s_law

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Head, Information Operations and Space Integration Branch
   PLI/PP&O/HQMC,
   Washington, DC

4. Dario Cadamuro
   NATO C3 Agency
   The Hague, Netherlands

5. Dan Boger
   Naval Postgraduate School
   Monterey, CA

6. Don Brutzman
   Naval Postgraduate School
   Monterey, CA

7. Terry Norbraten
   Naval Postgraduate School
   Monterey, CA

8. Don McGregor
   Naval Postgraduate School
   Monterey, CA

9. Paul Sandoz
   Sun Microsystems
   Palo Alto, CA

10. Santiago Percias-Geertsen
    Sun Microsystems
    Palo Alto, CA

11. Emil Serpa
    Sun Microsystems
    Palo Alto, CA

12. John Schneider
AgileDelta
Bellevue, WA

13. CAPT Scot Miller, USN
Naval Center for Tactical Systems Interoperability
San Diego, CA

14. Chris Gunderson
Naval Postgraduate School
Monterey, CA

15. Richard Lee
Office of the Secretary of Defense
Washington, D.C.

16. Erik Chaum
Naval Undersea Warfare Center
Newport, RI

17. Web3D Consortium
San Francisco, CA

18. Efficient XML Interchange
World Wide Web Consortium
Cambridge, MA

19. Mark Falash
Link Simulation and Training
Arlington, TX

20. Eytan Pollak
Link Simulation and Training
Arlington, TX

21. Mark Pullen
George Mason University
Fairfax, VA

22. Andreas Tolk
Old Dominion University
Norfolk, VA

23. Katherine Morse
AVP Technology, SAIC
San Diego, CA

24. Undersea Warfare XML List
Naval Postgraduate School
Monterey, CA

25. Undersea Warfare Announcements List
Naval Postgraduate School
Monterey, CA